TREBALL DE FI DE GRAU

Grau en Matemàtiques

Títol: Logaritme discret i aplicacions a la criptografia

Autor: Sergi Ferrer Fernàndez

Director: Jordi Quer Bosor

Departament: Matemàtica Aplicada II

Convocatòria: 2014 - 2015

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat de Matemàtiques i Estadística

Universitat Politècnica de Catalunya
Facultat de Matemàtiques i Estadística

Projecte final de carerra

Logaritme discret i aplicacions a la criptografia

Sergi Ferrer Fernàndez

Director: Jordi Quer Bosor

Departament de Matemàtica Aplicada II

Índex general

Introducció

Sigui G un grup cíclic finit d'ordre n, generat per un element g. L'aplicació $k \longmapsto g^k$ és un isomorfisme entre el grup $\mathbb{Z}/n\mathbb{Z}$ de les classes de congruència mòdul n i el grup G. Aquest isomorfisme s'anomena *exponencial discreta* de base g i el seu invers és el *logaritme discret*.

En el context de les matemàtiques, això són observacions trivials. Però quan es miren aquestes aplicacions des del punt de vista de l'algorísmica i la complexitat es plantegen problemes molt difícils de computar. Més precisament, l'exponencial discreta sempre es pot calcular de manera eficient amb un algorisme basat en la idea de dividir i vèncer; en canvi, no es coneixen algorismes eficients per a calcular logaritmes discrets en general, i es conjectura que, per a certs grups G, el problema és intractable.

El problema del logaritme discret ha assolit una gran importància per les seves aplicacions a la criptografia de clau pública, per construir protocols de xifrat i de firma digital. Actualment, els sistemes criptogràfics de clau pública que es fan servir basen la seva seguretat en la dificultat de resoldre un dels dos problemes següents: el problema de la factorització de nombres enters o el problema del logaritme discret, ja sigui en un subgrup del grup multiplicatiu d'un cos finit o en un subgrup del grup de punts d'una corba el·líptica sobre un cos finit. De fet, d'aquests mètodes, el que es considera més segur i el que recomanen tots els experts, és el tercer: el logaritme discret en corbes el·líptiques.

Així doncs, la seguretat de molts sistemes criptogràfics de clau pública es basa en la dificultat a l'hora de resoldre el logaritme discret. De fet, si es trobés un algorisme eficient per resoldre'l es comprometria la seguretat d'aquests sistemes criptogràfics i les comunicacions, per exemple a través d'Internet, deixarien de ser segures. Aquest fet, motiva l'estudi de la complexitat del logaritme discret.

L'objectiu principal d'aquest treball serà entendre bé el logaritme discret des del punt de vista computacional i estudiar els principals algorismes que es fan servir. Per a resoldre'l existeixen algorismes exponencials que es poden aplicar

per grups generals. Per al cas particular del grup multiplicatiu d'un cos finit es fa servir l'algorisme de càlcul d'índex, que és un algorisme no eficient però de complexitat subexponencial. Aquest algorisme no té un anàleg en el grup de punts d'una corba el·líptica, i és el responsable del fet que treballar en aquest segon grup sigui la millor opció ara per ara.

El contingut del treball es divideix en 5 capítols. El primer conté les definicions i resultats bàsics de matemàtiques, algorísmica i criptografia que ens seran útils més endavant en el treball. Els capítols 2 i 3 són el nucli del treball, on s'estudien els principals algorismes per a la resolució del logaritme discret: el capítol 2 està dedicat a algorismes per a grups generals, que són els millors algorismes per resoldre el logaritme discret sobre corbes el·líptiques, i el capítol 3 se centra en estudiar el càlcul d'índex, un algorisme subexponencial per al grup multiplicatiu d'un cos finit molt important ja que redueix significativament la complexitat respecte del logaritme sobre el grup de punts d'una corba el·líptica. En el quart ens centrem a comparar els dos problemes principals usats actualment en criptografia de clau pública: la factorització i el logaritme discret. Per acabar, el capítol 5 està dedicat a descriure la manera com el logaritme discret s'aplica a la pràctica explicant protocols criptogràfics concrets.

Capítol 1
Preliminars i conceptes bàsics

1. Dades binàries

Quan es parla de criptografia, automàticament es relaciona aquest concepte amb la informàtica. Això és degut a que les diferents tècniques criptogràfiques que es fan servir a la pràctica s'implementen amb ordinadors. Per tal que els ordinadors puguin interpretar la informació que reben és necessari treballar amb cadenes binàries de dades.

DEFINICIÓ 1.1. *Direm que B és una cadena binaria de longitud ℓ si:*

$$B \in \{0,1\} \times \overset{(\ell)}{\ldots\ldots} \times \{0,1\} = \{0,1\}^{\ell}.$$

DEFINICIÓ 1.2. *Es defineix el conjunt de cadenes binaries de mida arbitraria com:*

$$\{0,1\}^{*} = \bigcup_{\ell \geqslant 0} \{0,1\}^{\ell}.$$

2. Algorísmica

Abans de començar a parlar de criptografia convé introduir alguns conceptes algorísmics, ja que la criptografia té molta relació amb l'anàlisi i creació d'algorismes. Per exemple, quan es vol encriptar o desencriptar un missatge fan falta algorismes per fer-ho. També es fan servir algorismes per firmar documents digitals i acreditar-ne l'origen. A més, un adversari que intenti comprometre la seguretat del nostre sistema criptogràfic haurà d'utilitzar un algorisme per aconseguir-ho.

Diem que un algorisme és un conjunt de computacions que porten a la solució d'un problema. Més concretament, suposem que es vol solucionar un problema P definit amb unes dades (input $\mathcal{I}$). Un algorisme A per solucionar-lo és un conjunt de passos que transformen l'entrada $\mathcal{I}$ en una sortida $\mathcal{O}$, amb $\mathcal{I}, \mathcal{O} \in \{0,1\}^{*}$,

on la sortida ens dóna la solució del problema P:

$$\mathcal{I} \longrightarrow \boxed{\text{A}} \longrightarrow \mathcal{O}$$

Per exemple, considerant el problema P de trobar un camí eulerià en un graf, on l'entrada és el graf G. Un algorisme que resolgui el problema haurà de donar com a sortida un camí eulerià del graf o un NO si aquest no existeix.

Es poden comparar diferents algorismes que solucionen el mateix problema per decidir quin és millor. Per fer-ho es miren propietats com el temps d'execució i l'espai de memòria que ocupa l'algorisme en executar-se. Per definir aquestes propietats es farà servir la notació $O(f(n))$, $\theta(f(n))$ i $\Omega(f(n))$ amb la seva definició estàndard.

DEFINICIÓ 1.3. *Sigui $n \in \mathcal{I}$ una entrada de l'algorisme.*
Definim $\text{cost}(n)$ *com el nombre d'operacions bàsiques que fa l'algorisme (cada operació amb temps d'execució constant), en funció de l'entrada n.*
Definim $\text{cost}_E(n)$ *com la funció que compta els bits de memòria que ocupa l'algorisme per una entrada n.*

Aquestes funcions de cost es poden fitar per funcions $f(n)$ (com ara polinomis, exponencials, logaritmes, etc.) que donen una idea de la complexitat algorísmica. Notarem aquestes fites de la següent manera: $\text{cost}(n) = O(f(n))$.

Quan definim $\text{cost}(n)$ comptem les operacions que fa l'algorisme. Com que cada operació tarda un temps constant, $\text{cost}(n)$ és proporcional al temps d'execució de l'algorisme.

OBSERVACIÓ 1.1. La funció $\text{cost}(n)$ es defineix en funció de l'entrada $n \in \mathcal{I}$, però normalment quan es parla del cost d'un algorisme es fa en funció dels bits de l'entrada. Si l'entrada és un enter n, la seva mida és $m \approx \log_2(n)$ que són els bits que ocupa n. Direm que un algorisme amb $\text{cost}(n) = \log_2 n$ és lineal. En canvi, un algorisme amb $\text{cost}(n) = n$ és exponencial. Com es veurà a l'exemple 1.1, l'algorisme d'Euclides per calcular $\text{mcd}(a, b)$ és lineal en la mida de l'entrada.

OBSERVACIÓ 1.2. Quan es parla del cost d'un algorisme s'acostuma a diferenciar entre el cost en el cas pitjor, el cost en el cas mitjà i el cost en el cas millor. El cost en el cas pitjor és el cost màxim entre totes les entrades de mida n. El cost en el cas mitjà és el cost en el cas esperat, on cal fer servir probabilitats i esperances per calcular-lo. El cost en el cas millor és el cost mínim de totes les entrades de mida n. Si no es diu res, se suposarà que estem parlant del cost en el cas pitjor.

EXEMPLE 1.1. Veiem l'exemple de l'algorisme d'Euclides per calcular $\text{mcd}(a, b) = d$, on suposem $a > b$. En aquest cas, els passos de l'algorisme consisteixen en

realitzar una serie de divisions fins a obtenir el resultat del problema. A partir d'aquest resultat es pot construir la identitat de Bézout: trobar $x, y \in \mathbb{Z}$ tals que $ax + by = d$.

Definim $r_0 := a$ i $r_1 := b$. Llavors, dividim r_0 entre r_1 (i.e. $r_0 = r_1 q + r_2$) i ens guardem el residu r_2. Redefinim $r_0 := r_1$ i $r_1 := r_2$ i repetim el procés fins que $r_2 = 0$. Arribats a aquest punt, $\mathrm{mcd}(a, b) = r_1$.

Aquest algorisme té un cost en temps de $O(\log(a))$ tant en cas esperat com en cas pitjor. Veiem el següent exemple numèric $\mathrm{mcd}(28, 18)$:

$$\begin{aligned} r_0 = 28, &\quad r_1 = 18 \;\Rightarrow\; r_2 = 10, \\ r_0 = 18, &\quad r_1 = 10 \;\Rightarrow\; r_2 = 8, \\ r_0 = 10, &\quad r_1 = 8 \;\Rightarrow\; r_2 = 2, \\ r_0 = 8, &\quad r_1 = 2 \;\Rightarrow\; r_2 = 0. \end{aligned}$$

Per tant, $\mathrm{mcd}(28, 18) = 2$.

Tirant enrere les successives divisions de l'algorisme d'Euclides obtenim la identitat de Bézout ($28x + 18y = 2$). Per fer-ho s'aïllen els residus de les divisions $r_2 = r_0 - r_1 q$, fins a recuperar 28 i 18.

$$\begin{aligned} &2 = 10 - 1 \cdot 8, \\ &8 = 18 - 1 \cdot 10 \;\Rightarrow\; 2 = 10 - 1 \cdot (18 - 1 \cdot 10) = 2 \cdot 10 - 1 \cdot 18, \\ &10 = 28 - 1 \cdot 18 \;\Rightarrow\; 2 = 2 \cdot (28 - 1 \cdot 18) - 1 \cdot 18 = 2 \cdot 28 - 3 \cdot 18. \end{aligned}$$

Per tant, la identitat de Bézout és: $28 \cdot 2 - 18 \cdot 3 = 2$.

En aquest cas, el cost de l'algorisme no supera el nombre de bits de 28. Per altra banda, el cas millor té cost constant. Per comprovar-ho s'executa l'algorisme amb a múltiple de b, obtenint $\mathrm{mcd}(a, b) = b$ amb una sola divisió, prenent $r_0 = a$, $r_1 = b$ i veient que $r_2 = 0$.

DEFINICIÓ 1.4. *Complexitat algorísmica*

Es diu que un problema és tractable si hi ha un algorisme que el soluciona en temps polinòmic (i.e. $\mathrm{cost}(n) = O(f(n))$ *amb* $f(n)$ *un polinomi en* $\log n$*). Per contra, si no hi ha cap algorisme que solucioni el problema en temps polinòmic es diu que el problema és intractable.*

Els problemes intractables es poden classificar en molts tipus. En aquest treball tractarem amb problemes exponencials i subexponencials. Direm que el cost d'un algorisme és exponencial si $\mathrm{cost}(n) = \Omega(a^{\log n})$ *per* $a > 1$*. Direm que el cost d'un algorisme és subexponencial si es troba per sota de tota exponencial però per sobre de tot polinomi.*

OBSERVACIÓ 1.3. Els algorismes per calcular el logaritme discret treballen amb enters. Per obtenir-ne el cost es pot considerar constant $\theta(1)$ el cost de les operacions bàsiques amb nombres enters (sumar, multiplicar, consultar una posició de memòria, guardar un valor, etc.) i, així, la quantitat total d'aquestes operacions durant l'execució ens dóna el cost de l'algorisme.

Una altra opció és considerar amb cost constant les operacions per un sol bit o un nombre de bits fixat. Així doncs, si per exemple tenim dos enters de t-bits el cost d'una suma seria $t \cdot \theta(1)$ i el d'una multiplicació amb un algorisme naïf seria $t^2 \cdot \theta(1)$. En aquest cas, per calcular el cost total d'un algorisme hauríem d'especificar com fer les operacions bàsiques, definint un algorisme per cada operació. Per exemple, per a la multiplicació d'enters hi ha diversos algorismes cada un amb diferent cost.

Els sistemes criptogràfics que s'explicaran basen la seva seguretat en el fet que el problema del logaritme discret sigui intractable. Com que els algorismes per dur a terme les operacions bàsiques són polinòmics i afegir un factor polinòmic no modifica un problema de tractable a intractable, procedirem a calcular el cost dels algorismes segons la primera opció explicada.

3. Criptografia

La **criptografia** és el conjunt de tècniques que s'utilitzen per garantir la seguretat de les comunicacions i de l'emmagatzematge de dades. Aquestes dades s'anomenen missatges i es transformen en textos xifrats fent servir un algorisme que usa una cadena binaria aleatòria: la clau. Per tant, l'espai en el qual treballem està format per missatges, claus i textos xifrats.

DEFINICIÓ 1.5. *Definim un espai criptogràfic com un triplet* $(\mathcal{K}, \mathcal{M}, \mathcal{C})$ *complint que:*

$$\mathcal{K} = \{0,1\}^{\ell}, \quad \mathcal{M} = \{0,1\}^{r}, \quad \mathcal{C} = \{0,1\}^{s},$$

amb $\mathcal{K}$ *el conjunt de claus,* $\mathcal{M}$ *el conjunt de missatges i* $\mathcal{C}$ *el conjunt de codis xifrats. Normalment, es té que* $r = s$ *i* ℓ *és més petit. Els missatges de mida més gran que* r *es separen en blocs d'aquest tamany.*

Pel cas de la firma digital, que s'explicarà més endavant, s'ha d'adaptar l'espai criptogràfic. Quan tractem amb la firma digital no es parla del conjunt de textos xifrats $\mathcal{C}$ *sinó del conjunt de firmes* $\mathcal{S}$*.*

Direm que un sistema criptogràfic és segur si es compleixen les següents propietats: la confidencialitat del missatge, la integritat de les dades, l'autenticitat i el no-rebuig.

La **confidencialitat** o **seguretat** de les dades és la propietat que garanteix que

el missatge es manté en secret. És a dir, una persona no autoritzada (sense la clau) no ha de poder llegir el missatge i, més encara, no ha de poder obtenir-ne cap informació. Per assegurar la confidencialitat es fan servir algorismes d'encriptació i desencriptació (E, D).

Hi ha diferents definicions de seguretat de les dades segons l'objectiu que es pretengui aconseguir. Per exemple, per tal que un sistema tingui seguretat perfecta cal que la probabilitat d'obtenir un text xifrat c (si triem k aleatòria) sigui la mateixa per a qualsevol missatge m. El problema d'aquesta definició és que la clau ha de tenir com a mínim la mida del missatge (Teorema de seguretat perfecta de Shanon). Una definició de seguretat més útil a la pràctica és la seguretat semàntica on donat un xifrat c i dos missatges m_0 i m_1 es demana que no es pugui distingir si c és el xifrat de m_0 o de m_1.

La **integritat** s'ocupa de garantir que el missatge no ha estat modificat durant la comunicació o detectar si ho ha sigut. Per aconseguir aquesta propietat es fan servir funcions de hash criptogràfiques.

L'**autenticitat** permet garantir qui és l'emissor del missatge, si aquest així ho vol certificar. El **no-rebuig** és la propietat que assegura no poder negar haver firmat un document electrònicament. Per aconseguir aquestes propietats es fa servir la firma digital.

Cal esmentar la importància de la firma digital i el fet que compleixi les propietats d'autenticitat i no-rebuig. Sobretot a partir del 2003, any que es va publicar la Llei 59/2003 on s'estableix que la firma digital té la mateixa validesa legal que la firma manuscrita. Si no es complissin les dues propietats esmentades, es podria falsificar la firma o negar haver firmat un document electrònicament i la firma digital perdria tota la seva validesa.

Com hem vist, la criptografia serveix per xifrar i desxifrar fitxers i dades. Això permet mantenir comunicacions segures, per exemple en comunicacions client-servidor a través d'Internet. També permet saber la procedència d'un missatge usant la firma digital i assegurar que un missatge no s'ha modificat amb funcions de hash. Així doncs, amb el desenvolupament actual de la informàtica i Internet la criptografia és de vital importància.

3.1. Tipus de criptografia.

Per entendre millor com funciona la criptografia cal conèixer els diferents tipus de sistemes criptogràfics que existeixen. Podem classificar els sistemes criptogràfics en dos tipus segons la clau utilitzada: els sistemes de clau secreta i els sistemes de clau pública.

Sigui $(\mathcal{K}, \mathcal{M}, \mathcal{C})$ un espai criptogràfic. La diferència entre els dos tipus de criptografia radica en la forma de definir i obtenir la clau $k \in \mathcal{K}$.

La **criptografia de clau privada**, de **clau secreta** o **criptografia simètrica** es fa servir sobretot per encriptar, ja que resulta menys costós que fer-ho amb criptografia asimètrica. En aquests sistemes, la clau k s'utilitza tant en l'algorisme d'encriptació com en el de desencriptació i la coneixen l'emissor i els receptors (persones autoritzades a llegir el missatge).

Es suposa que es volen comunicar dues persones A i B, on les dues coneixen k. Per fer-ho procedeixen de la següent manera. L'emissor A encripta el missatge usant la clau k i li envia al receptor B. Si algú intercepta aquest missatge xifrat, no serà capaç de desxifrar-lo sense la clau. Un cop el missatge encriptat arriba a B només ha d'usar k per desencriptar-lo i obtenir el missatge original.

En la **criptografia de clau pública** o **criptografia asimètrica**, a diferència d'en criptografia simètrica, el conjunt de claus $\mathcal{K}$ consta de dues components $\mathcal{K}_s, \mathcal{K}_p = \{0,1\}^\ell$. En aquest tipus de criptografia, una clau és de la forma $k = (k_s,\ k_p)$ amb k_p la component pública, que pot conèixer qualsevol persona, i k_s la component privada, que només coneix la persona que genera la clau.

Les components k_s i k_p de la clau estan lligades per una funció unidireccional γ. Aquesta funció utilitza alguna propietat matemàtica que ens permeti obtenir de forma eficient (temps polinòmic) la clau pública a partir de la privada ($\gamma(k_s) = k_p$). Però, en canvi, el pas invers ($\gamma^{-1}(k_p) = k_s$) sigui molt més complicat (exponencial o subexponencial). Gràcies a aquesta asimetria podem exposar públicament k_p tenint la certesa que ningú aconseguirà conèixer k_s.

DEFINICIÓ 1.6. *Definim una funció unidireccional com:*

$$\begin{array}{rccc} \gamma: & \mathcal{K}_s & \longrightarrow & \mathcal{K}_p \\ & k_s & \longmapsto & \gamma(k_s) = k_p \end{array}$$

tal que hi ha un algorisme que computa γ en temps polinòmic i que donat $y \in \mathcal{K}_p$ és intractable trobar $x \in \mathcal{K}_s$ tal que $\gamma(x) = y$.

Per exemple, es conjectura que l'exponencial discreta, que té per invers el logaritme discret, és una funció unidireccional. També hi ha funcions unidireccionals basades en la conjectura que la factorització d'enters és un problema intractable. Aquestes últimes funcions s'utilitzen en l'algorisme RSA. En aquest treball ens centrarem a estudiar les propietats del logaritme discret i veurem els algorismes més eficients que es coneixen per calcular-lo. Per últim, veurem com s'aplica el logaritme discret per crear protocols criptogràfics.

Entre les aplicacions de la criptografia asimètrica en destaquen sobretot dues.

La primera aplicació consisteix en encriptar i desencriptar, que s'acostuma a fer servir per transmetre la clau k necessària per comunicar-se via criptografia simètrica. L'altra aplicació destacada de la criptografia asimètrica és la firma digital, que només es pot dur a terme amb aquest tipus de criptografia.

3.2. Tècniques criptogràfiques.

Sistemes d'encriptat

Sigui $(\mathcal{K}, \mathcal{M}, \mathcal{C})$ un espai criptogràfic. Un algorisme d'encriptació E transforma els missatges en codis xifrats i un algorisme de desencriptació D recupera el missatge original a partir del codi xifrat. Veiem la definició d'aquests algorismes (E, D).

Definició 1.7. *Definim un algorisme d'encriptació (E) i un algorisme de desencriptació (D) de la forma següent:*

$$\begin{array}{rccc} E: & \mathcal{K}\times\mathcal{M} & \longrightarrow & \mathcal{C} \\ & (k,m) & \longmapsto & E(k,m)=c \end{array}$$

$$\begin{array}{rccc} D: & \mathcal{K}\times\mathcal{C} & \longrightarrow & \mathcal{M} \\ & (k,c) & \longmapsto & D(k,c)=m \end{array}$$

on s'ha de complir l'equació de consistència: $D(k, E(k, m)) = m$.

La criptografia moderna segueix la màxima de Shannon (una reformulació dels principis de Kerckhoffs) que diu "L'adversari coneix el sistema". Amb això es suposa que els algorismes d'encriptació i desencriptació són públics i l'únic que cal mantenir en secret és la clau, no pas el funcionament sencer del sistema.

Exemple 1.2. Veiem com a exemple l'algorisme DES (Data Encription Standard). Per a aquest protocol de criptografia simètrica expliquem les operacions que fan E i D per obtenir el codi xifrat i per recuperar el missatge original.

Aquest algorisme funciona per blocs, és a dir, el missatge original es separa en blocs (de 64-bits) i els algorismes d'encriptació i desencriptació s'apliquen a cadascun dels blocs. Els algorismes E i D consten de 3 passos:

1. Agafem el bloc o missatge m i en permutem els bits, amb una permutació P.
2. Apliquem una xarxa de Feisel (Feisel Network) al resultat obtingut al pas 1.
3. Realitzem la permutació inversa P^{-1} al resultat de 2.

En la xarxa de Feisel es fan una serie de transformacions en els bits del missatge, com ara permutar-los, sumar bits mòdul 2, combinar els bits de la clau i del missatge, etc.

Cal fer notar la diferencia a l'hora d'encriptar i desencriptar missatges usant criptografia simètrica o criptografia asimètrica. En criptografia asimètrica s'encripta amb la component pública de la clau, de manera que tothom pot encriptar, i es desencripta amb la component secreta, de manera que només pot desxifrar el missatge la persona que genera la clau. Veiem, doncs, les diferencies a la taula següent:

Criptografia simètrica	Criptografia asimètrica
$E: \begin{array}{ccc} \mathcal{K}\times\mathcal{M} & \longrightarrow & \mathcal{C} \\ (k,m) & \longmapsto & E(k,m)=c \end{array}$ $D: \begin{array}{ccc} \mathcal{K}\times\mathcal{C} & \longrightarrow & \mathcal{M} \\ (k,c) & \longmapsto & D(k,c)=m \end{array}$	$E: \begin{array}{ccc} \mathcal{K}_p\times\mathcal{M} & \longrightarrow & \mathcal{C} \\ (k_p,m) & \longmapsto & E(k_p,m)=c \end{array}$ $D: \begin{array}{ccc} \mathcal{K}_s\times\mathcal{C} & \longrightarrow & \mathcal{M} \\ (k_s,c) & \longmapsto & D(k_s,c)=m \end{array}$
Equació de consistència	Equació de consistència
$D(k,E(k,m))=m$	$D(k_s,E(\gamma(k_s),m))=m$

A continuació veiem com s'encripta i desencripta fent servir criptografia de clau pública. Com ja s'ha mencionat, per encriptar és millor fer servir criptografia simètrica. Per això cal que les dues persones que es volen comunicar coneguin la clau k. Per tant, primer s'ha de transmetre aquesta clau amb confidencialitat, i és aquí on s'usa criptografia asimètrica.

Siguin A i B dues persones que es volen comunicar. Llavors, per passar-se la clau k procedeixen de la següent manera:

1. A genera la clau aleatòria k de criptografia simètrica.
2. B crea una clau (k_s, k_p) i fa pública la component k_p.
3. A usa la clau k_p per encriptar el missatge $m = k$ i li envia el missatge encriptat a B.
4. B només ha de desencriptar el missatge rebut amb k_s per obtenir k.

Finalment, A i B posseeixen la clau k i ja es poden comunicar via criptografia simètrica. Observem que aquest protocol té un problema de seguretat en el pas 3, ja que B no pot saber si el missatge prové de A o no. Per solucionar aquest problema A pot firmar el codi xifrat que li envia a B. Per fer-ho usa un altre parell de claus (k_s, k_p) com s'explica en el següent apartat.

La firma digital

Per crear la firma digital cal modificar l'espai criptogràfic sobre el qual treballem.

DEFINICIÓ 1.8. *Definim un espai criptogràfic com un triplet* $(\mathcal{K}, \mathcal{M}, \mathcal{S})$ *complint que:*

$$\mathcal{K} = \{(k_s, k_p) \in \mathcal{K}_s \times \mathcal{K}_p \mid k_p = \gamma(k_s)\}, \quad \mathcal{M} = \{0,1\}^r, \quad \mathcal{S} = \{0,1\}^s,$$

amb $\mathcal{K}$ *el conjunt de claus i* $\mathcal{K}_s, \mathcal{K}_p = \{0,1\}^\ell$*,* $\mathcal{M}$ *el conjunt de missatges i* $\mathcal{S}$ *el conjunt de firmes.*

Una firma digital és un annex que s'afegeix al missatge per certificar qui és l'emissor. Per fer-ho, es fa servir criptografia de clau pública i s'aprofita que el firmant és l'única persona que coneix la component secreta k_s de la clau. Veiem una definició formal de la firma digital, on tenim un algorisme per generar la firma i un algorisme per verificar-la (F, V).

DEFINICIÓ 1.9. *Sigui* $(\mathcal{K}, \mathcal{M}, \mathcal{S})$ *un espai criptogràfic. Definim un algorisme de generació de la firma* (F) *i un algorisme de verificació de la firma* (V) *de la forma següent:*

$$\begin{array}{rccc} F: & \mathcal{K}_s \times \mathcal{M} & \longrightarrow & \mathcal{S} \\ & (k_s, m) & \longmapsto & F(k_s, m) = \mathrm{f} \end{array}$$

$$\begin{array}{rccc} V: & \mathcal{K}_p \times \mathcal{M} \times \mathcal{S} & \longrightarrow & \{0,1\} \\ & (k_p, m, \mathrm{f}) & \longmapsto & V(k_p, m, \mathrm{f}) \end{array}$$

si l'algorisme de verificació dóna 0 es rebutja la firma i si l'algorisme de verificació dóna 1 s'accepta la firma.

S'ha de complir l'equació de consistència $V(\gamma(k_s), m, F(k_s, m)) = 1$. A més, si s'intenta verificar usant un missatge o una firma incorrecta, en general l'algorisme donarà 0.

Sigui A un firmant que vol enviar un missatge firmat a un receptor B. El procediment general que es segueix per crear la firma digital i autentificar la procedència del missatge és el següent:

1. A té una clau asimètrica (k_p, k_s), on la component k_p és pública i coneguda per B.
2. A firma el missatge amb la clau secreta $F(k_s, m) = \mathrm{f}$ i li envia (m, f) a B.
3. B només ha de verificar la firma rebuda amb k_p, veient que $V(k_p, m, \mathrm{f}) = 1$.

Al verificar la firma s'assegura que el missatge rebut procedeix de A, ja que és l'únic que coneix k_s i, per tant, l'únic capaç de generar la firma. Per crear la firma digital s'acostuma a usar el hash criptogràfic del missatge h en comptes del missatge complet m, ja que h té una mida menor i fa menys costós generar

la firma.

Funcions de hash criptogràfiques

Una funció de hash criptogràfica envia un missatge de mida arbitrària a una cadena binaria de mida petita ℓ fixada, sense necessitat de cap clau per fer-ho.

DEFINICIÓ 1.10. *Definim una funció de hash criptogràfica com:*

$$\begin{array}{rccc} h: & \{0,1\}^* & \longrightarrow & \{0,1\}^\ell \\ & m & \longmapsto & h(m) \end{array}$$

Complint que:

1. *Computació fàcil: $h(m)$ es pot calcular en temps polinòmic.*
2. *Antiimatge resistent (1): Donat $x \in \{0,1\}^\ell$ és intractable trobar $m \in \{0,1\}^*$ tal que $h(m) = x$.*
3. *Antiimatge resistent (2): Donat $m \in \{0,1\}^*$ és intractable trobar $m' \neq m$ tal que $h(m') = h(m)$.*
4. *Resistència a col·lisió: És intractable trobar $m_1, m_2 \in \{0,1\}^*$ diferents tals que $h(m_1) = h(m_2)$.*

Un exemple de funció de hash criptogràfica és l'algorisme del SHA-1 explicat a l'apèndix 1 (Algorisme A.1).

Les funcions de hash criptogràfiques ens ajuden a garantir la integritat de m. Per fer-ho apliquem la funció de hash al missatge i afegim el resultat al final obtenint $\overline{m} = (m, h(m))$. És suficient amb garantir la integritat de $h(m) = h$, ja que donat (m', h) es pot detectar que s'ha canviat m per m' veient que $h(m') \neq h$.

Fa falta garantir la integritat de h d'alguna forma, sinó es podria canviar m per m' modificant també el hash del missatge $(m', h(m'))$. Podem procedir de dos maneres per garantir la integritat del hash. Un procediment consisteix en prendre el parell $\overline{m} = (m, h(m))$ i encriptar-lo $E(k, \overline{m}) = c$. Un segon procediment, consisteix en prendre el mateix parell $\overline{m}$ i firmar $h(m)$ amb la firma digital. Observem que firmar únicament el hash té molt menys cost que firmar el missatge original.

Si suposem que enviem un missatge (m, h) on el hash h no pot ser modificat, necessitem que la nostra funció compleixi les propietats de la definició de hash criptogràfic per assegurar que el missatge tampoc serà modificat. Sinó es podria trobar fàcilment un missatge m' tal que (m', h) és un parell vàlid (i.e. $h(m') = h$), on hem aconseguit modificar el missatge tot i garantir la integritat de h.

Donada una funció de hash criptogràfica l'objectiu teòric és arribar a una fita inferior del cost de trobar una antiimatge o una col·lisió. Malauradament, aquestes fites són complicades de trobar i el que s'acostuma a fer és mirar la complexitat de l'atac més eficient que es coneix obtenint una fita superior del cost.

4. El logaritme discret

Sigui G un grup cíclic d'ordre n i g un generador. Definirem el logaritme i l'exponencial discreta com dues aplicacions bijectives entre els grups G i $\mathbb{Z}/n\mathbb{Z}$, inverses una de l'altra.

DEFINICIÓ 1.11. *Exponencial discreta*

Sigui G grup cíclic i g un generador. Definim l'exponencial discreta en base g com:

$$\begin{array}{rccc} \exp_g: & \mathbb{Z}/n\mathbb{Z} & \longrightarrow & G \\ & k & \longmapsto & g^k \end{array}$$

Algorisme per calcular l'exponencial discreta

L'algorisme que es fa servir per calcular l'exponencial discreta es basa en la tècnica de dividir i vèncer. Es vol calcular g^k i per fer-ho es calcula una exponencial de la meitat de tamany:

$$h = \begin{cases} g^{k/2} & \text{Per } k \text{ parell,} \\ g^{(k-1)/2} & \text{Per } k \text{ senar.} \end{cases}$$

Un cop fet això, per arribar a trobar g^k només cal elevar al quadrat el resultat obtingut h:

$$g^k = \begin{cases} h^2 & \text{Per } k \text{ parell,} \\ h^2 \cdot g & \text{Per } k \text{ senar.} \end{cases}$$

Aquest procediment es repeteix recursivament fins a arribar al cas base, amb $k = 1$ i on el resultat és g. Veiem un exemple d'aquest algorisme a continuació.

EXEMPLE 1.3. Calcular 7^{1345}

$7^{1345} = 7^{672} \cdot 7^{672} \cdot 7 \quad \leftarrow \quad 7^{672} = 7^{336} \cdot 7^{336} \quad \leftarrow \quad 7^{336} = 7^{168} \cdot 7^{168} \quad \leftarrow \quad 7^{168} = 7^{84} \cdot 7^{84} \quad \leftarrow \quad 7^{84} = 7^{42} \cdot 7^{42} \quad \leftarrow \quad 7^{42} = 7^{21} \cdot 7^{21} \quad \leftarrow \quad 7^{21} = 7^{10} \cdot 7^{10} \cdot 7 \quad \leftarrow \quad 7^{10} = 7^5 \cdot 7^5 \quad \leftarrow \quad 7^5 = 7^2 \cdot 7^2 \cdot 7 \quad \leftarrow \quad 7^2 = 7 \cdot 7$

Cost de l'algorisme

PROPOSICIÓ 1.1. *El cost de l'algorisme de càlcul de l'exponencial discreta és* $\text{cost}(k) = O(\log(k))$.

Demostració.

Notem el cost de l'algorisme per calcular g^k com $C(k)$. Llavors, $C(k)$ compleix la següent recurrència:

$$C(k) \;=\; C(\lfloor k/2 \rfloor) \;+\; O(1).$$

Apliquem la recurrència repetidament:

$$\begin{aligned} C(k) \;&=\; C(\lfloor k/2 \rfloor) \;+\; O(1) \;=\; C(\lfloor k/4 \rfloor) \;+\; O(1) \;+\; O(1) \;=\; \ldots \;= \\ &=\; \sum_{m=1}^{\log_2(k)} O(1) \;=\; O(\log_2(k)). \end{aligned}$$

□

Definició i propietats del Logaritme discret

Definició 1.12. *Logaritme discret*

Sigui G un grup cíclic i g un generador. Definim el logaritme discret en base g com la inversa de l'exponencial discreta. Notem com $\log_g$ o ind_g:

$$\begin{aligned} \log_g : \;\; G \;&\longrightarrow\; \mathbb{Z}/n\mathbb{Z} \\ g^k \;&\longmapsto\; \log_g(g^k) = k \end{aligned}$$

El logaritme discret conserva moltes de les propietats bàsiques del logaritme ordinari en $\mathbb{R}$.

Propietat 1.2. *La funció logaritme discret* $(\log_g)$ *compleix:*

1. $\log_g(a \cdot b) \;=\; \log_g(a) \;+\; \log_g(b)$.
2. $\log_g(a^t) \;=\; t \log_g(a) \quad \forall\, t \in \mathbb{Z}$.
3. *Si r és un generador de G (i.e. $\langle g \rangle \;=\; \langle r \rangle$) llavors* $\log_g(h) \;=\; \log_g(r) \log_r(h)$.

Demostració.

(1) Com $a, b \in G$ llavors $\exists\; k_1,\; k_2$ tals que $a \;=\; g^{k_1},\; b \;=\; g^{k_2}$.
Per tant,

$$\log_g(a \cdot b) \;=\; \log_g(g^{k_1} \cdot g^{k_2}) \;=\; \log_g(g^{k_1+k_2}) \;=\; k_1 \;+\; k_2,$$

$$\log_g(a) \;+\; \log_g(b) \;=\; \log_g(g^{k_1}) \;+\; \log_g(g^{k_2}) \;=\; k_1 \;+\; k_2.$$

(2) Escrivim $a \;=\; g^k$ com abans i, per tant, $\log_g(a) \;=\; k$.

$$\log_g(a^t) \;=\; \log_g((g^k)^t) \;=\; \log_g(g^{t \cdot k}) \;=\; t \cdot k \;=\; t \log_g(a).$$

(3) Com que r i g són generadors de G podem escriure $h \in G$ en les dues bases. És a dir, $h = g^k = r^t$. A més, r es pot escriure en base g com $r = g^s$.
Per tant,

$$g^k = r^t = (g^s)^t = g^{st}, \quad on \;\; k = \log_g(h),\; t = \log_r(h),\; s = \log_g(r).$$

□

La propietat 1.2.3 ens dóna la fórmula per canviar la base del logaritme discret. La fórmula obtinguda és la mateixa que obtenim en el logaritme ordinari. Cal recordar que només és possible usar aquesta propietat per generadors del grup cíclic. Donat g un generador del grup G, els generadors del grup són tots els $r = g^k$ tal que $\text{mcd}(k, n) = 1$. Aquest fet ens assegura que $\log_g(r) = k$ és invertible en $\mathbb{Z}/n\mathbb{Z}$. Per tant, si sabem calcular el logaritme en una base g, el podem calcular en qualsevol base r amb la fórmula:

$$\log_r(h) = \frac{\log_g(h)}{\log_g(r)}.$$

El logaritme discret es defineix de forma equivalent si el grup G s'expressa en notació additiva, on la multiplicació es substitueix per una suma i elevar a k passa a ser multiplicar per k. Aquest és el cas, per exemple, del logaritme discret sobre corbes el·líptiques.

Definició 1.13. *Logaritme discret (notació additiva)*

Sigui G un grup cíclic i P un generador del grup. Definim el logaritme discret en base P com la inversa de l'exponencial discreta. Notem com $\log_P$ o ind_P:

$$\begin{aligned} \log_P : \; G &\longrightarrow \mathbb{Z}/n\mathbb{Z} \\ kP &\longmapsto \log_P(kP) = k \end{aligned}$$

Sigui G un grup finit. Observem que si el grup G no és cíclic, la funció exponencial discreta no és bijectiva, ja que la imatge de l'exponencial és $\langle g \rangle \subsetneq G$. En aquest cas, el logaritme discret no està definit. Per evitar això, definim l'exponencial i el logaritme sobre un subgrup cíclic de G. Per generar el subgrup, s'acostuma a prendre un element g d'ordre n i es crea el subgrup generat per g. Notem aquest grup com $\langle g \rangle$. Per tant, podem suposar $G = \langle g \rangle$ cíclic.

5. Grups cíclics

Sigui G un grup cíclic i g un generador. Per $h \in G$ direm que l'ordre de h és $\text{ord}(h) = \min\{k \mid h^k = 1\}$. En particular, $\text{ord}(g) = |G| = n$.

Com ja hem vist, les funcions logaritme discret ($\log_g$) i exponencial discreta ($\exp_g$) en base g són aplicacions entre els grups $\mathbb{Z}/n\mathbb{Z}$ i G. En aquesta secció s'explicaran els grups G més habituals per generar aquestes funcions.

Un cop definit el grup G, el següent pas serà escriure els elements del grup com a cadenes binàries per tal que puguin ser interpretats per un ordinador:

$$G \hookrightarrow \{0,1\}^{\ell} \tag{1}$$

Segons la representació triada pot variar l'espai de memòria necessari per guardar els elements del grup. Tot i així, a vegades és convenient ocupar una mica més de memòria de forma que la representació dels elements faciliti l'execució dels algorismes.

La representació en binari defineix un ordre per al grup G, associat a l'ordre habitual dels nombres enters. Donats $a, b \in G$, aquest ordre permet comprovar fàcilment si $a < b$ o $b < a$.

Sigui $\mathbb{A}$ un anell. Notem $\mathbb{A}^*$ el conjunt d'elements invertibles de $\mathbb{A}$. El següent teorema, estàndard d'àlgebra, ens serà útil per definir grups G sobre els que aplicar el logaritme discret.

TEOREMA 1.3. *Sigui $\mathbb{K}$ un cos i $(\mathbb{K}^*, \cdot)$ el seu grup multiplicatiu.*
Donat $G \subseteq \mathbb{K}^*$ un subgrup finit. Aleshores, G és un grup cíclic.

Observem que si $\mathbb{K}$ és un cos finit de q elements llavors el grup multiplicatiu $\mathbb{K}^*$ té $n = q - 1$ elements.

5.1. Grup multiplicatiu d'un cos finit.

El grup $\mathbb{F}_p^*$

Definim el cos de p elements com $\mathbb{F}_p = \mathbb{Z}/p\mathbb{Z}$. Els elements del grup $\mathbb{F}_p^* = \{1, 2, ..., p-1\}$ són els residus de la divisió entera per p. Observem que $\mathbb{F}_p^*$ és cíclic (Teorema 1.3).

Per treballar amb les operacions del cos $\mathbb{F}_p$ es fa servir la suma i la multiplicació habitual en $\mathbb{Z}$, es divideix el resultat per p i es pren el residu. Notarem aquestes operacions per $a, b \in \mathbb{F}_p^*$ com: $a + b \pmod p$ i $a \cdot b \pmod p$.

Tots els elements de $\mathbb{F}_p^*$ són invertibles per definició. Per $a \in (\mathbb{Z}/p\mathbb{Z})^*$ es compleix que $\text{mcd}(a, p) = 1$. Per invertir a s'aplica l'algorisme d'Euclides explicat en l'exemple 1.1. Amb aquest algorisme es calcula la identitat de Bézout $ax + py = 1$. Llavors, es té que $a^{-1} = x \pmod p$.

Tal com s'afirma en (1) cal escriure els elements del grup en binari per introduir-los en un ordinador. Per fer-ho representem els elements de $\mathbb{Z}/p\mathbb{Z}$ en base 2.

El grup $\mathbb{F}_q^*$

Definim el cos de $q = p^t$ elements com $\mathbb{F}_q = \mathbb{F}_p[x]/\langle f\rangle$ on $f(x) \in \mathbb{F}_p[x]$ és un polinomi primer de grau t. Pel teorema 1.4, estàndard d'estructures algebraiques, existeix un cos de $q = p^t$ elements per qualsevol $t \in \mathbb{N}$. A més, el teorema 1.3 ens assegura que el grup $\mathbb{F}_q^*$ és cíclic.

TEOREMA 1.4. *Sigui $\mathbb{F}_p$ un cos finit. Per a cada natural t existeix un polinomi primer $f(x) \in \mathbb{F}_p[x]$ de grau t.*

Els elements del grup $\mathbb{F}_q^*$ són els polinomis de grau menors que $t = \deg(f)$. On donat $p(x) \in \mathbb{F}_p[x]$, es divideix el polinomi $p(x)$ per $f(x)$ i ens quedem amb el residu.

En les operacions del cos es fan servir la suma, la multiplicació i la divisió euclidiana de polinomis. Prenem dos polinomis $y(x), z(x) \in \mathbb{F}_q = \mathbb{F}_p[x]/\langle f\rangle$. Per la suma només cal sumar els coeficients dels polinomis mòdul p. Com que la suma no fa augmentar el grau del polinomi, amb això és suficient. Per la multiplicació es procedeix a fer la multiplicació habitual de polinomis i els coeficients del resultat s'agafen mòdul p. Com que el grau haurà augmentat, cal dividir el polinomi obtingut per $f(x)$ i quedar-se amb el residu.

Donat un polinomi $p(x) \in \mathbb{F}_q^*$ es compleix que $\text{mcd}(p(x), f(x)) = 1$. Per invertir $p(x)$ es procedeix de forma equivalent que en l'algorisme d'Euclides per enters explicat en l'exemple 1.1. Aquest algorisme permet calcular la identitat de Bézout $p(x)\alpha(x) + f(x)\beta(x) = 1$. Llavors, es té que $p^{-1}(x) = \alpha(x)$.

Per definir la transformació dels elements del grup per introduir-los a l'ordinador es procedeix de la següent forma. Sabem que els elements del grup són de la forma $a_0+a_1X+a_2X^2+...+a_{t-1}X^{t-1}$. Per tant, es pren la tupla $(a_0, a_1, a_2, ..., a_{t-1})$ on cada $a_i \in \mathbb{F}_p$ i es passa cada $0 \leqslant a_i < p$ a base 2. D'aquesta manera és senzill operar amb polinomis, ja que en tenim els seus coeficients.

5.2. El grup de les corbes el·líptiques $E(\mathbb{F}_q)$.

Una corba el·líptica sobre un cos finit $\mathbb{F}_q$, amb $q = p^t$, es defineix usant equacions de Weierstrass. Una equació de Weierstrass, en general, és una equació de la forma:

$$Y^2 + a_1XY + a_3Y = X^3 + a_2X^2 + a_4X + a_6 \quad a_i \in \mathbb{F}_q. \tag{2}$$

Si la característica del cos és diferent de 2, es pot simplificar l'equació amb el canvi de variables $Y = \widetilde{Y} - \frac{a_1}{2}X - \frac{a_3}{2}$ obtenint:

$$\widetilde{Y}^2 = X^3 + \widetilde{a}_2X^2 + \widetilde{a}_4X + \widetilde{a}_6 \quad \widetilde{a}_i \in \mathbb{F}_q. \tag{3}$$

Si, a més, la característica del cos és diferent de 3 es pot aplicar el canvi de variables $X = \widetilde{X} - \frac{\widetilde{a}_2}{3}$ obtenint:

$$\widetilde{Y}^2 = \widetilde{X}^3 + a\widetilde{X} + b \quad a, b \in \mathbb{F}_q. \tag{4}$$

Els espais afins i els espais projectius estan estretament relacionats. Un espai afí es pot entendre com un espai projectiu al qual se li treu un hiperplà, l'hiperplà de l'infinit. Una corba el·líptica és una corba de l'espai afí a la que se li afegeix el punt de l'infinit P_∞. Definim el conjunt de punts d'una corba el·líptica sobre el cos $K = \mathbb{F}_q$ com:

$$E(K) = \{(x, y) \in K^2 \mid Y^2 + a_1XY + a_3Y = X^3 + a_2X^2 + a_4X + a_6\} \cup \{P_\infty\}. \tag{5}$$

Aquesta corba es pot passar a l'espai projectiu homogeneïtzant. Llavors, els punts de la corba, punts projectius a $\mathbb{P}^2(K)$, són les classes d'equivalència dels triplets $(x, y, z) \in K^3$. On dos triplets són equivalents si un és múltiple de l'altre amb coeficient diferent de zero. Es denota la classe d'equivalència de (x, y, z) com $[x : y : z]$. Per tant, el conjunt de punts de la corba el·líptica queda definit de la següent manera:

$$E(K) = \{[x : y : z] \in \mathbb{P}^2(K) \mid y^2z + a_1xyz + a_3yz^2 = x^3 + a_2x^2z + a_4xz^2 + a_6z^3\}.$$

L'únic punt on $z \neq 1$ és el punt de l'infinit $P_\infty = [0 : 1 : 0]$, que pertany a la corba el·líptica independentment dels paràmetres de l'equació.

Usant la definició de corba el·líptica (5) és fàcil veure que:

$$\#\text{ punts de } E(\mathbb{F}_q) \leqslant 2q + 1.$$

Considerem les parelles de punts (x, y) amb $x, y \in \mathbb{F}_q$ i y solució de l'equació de segon grau (5) per x fixada. Com a molt tenim dues solucions per cada x obtenint un total de $2q$ punts, que juntament amb el punt de l'infinit dóna la fita establerta.

Una millor fita per al nombre de punts d'una corba el·líptica ens la dóna el Teorema de Hasse (pàg 174 de [**1**]).

TEOREMA 1.5 (Hasse). *Sigui $E(\mathbb{F}_q)$ una corba el·líptica. Llavors:*

$$q + 1 - 2\sqrt{q} \leqslant \#\text{ punts de } E(\mathbb{F}_q) \leqslant q + 1 + 2\sqrt{q}.$$

Es pot definir una operació en una corba el·líptica per donar-li estructura de grup, gràcies a propietats de la geometria algebraica. Per aquest grup s'acostuma a usar la notació additiva en comptes de la multiplicativa, on el logaritme discret s'expressa de forma diferent (Definició 1.13).

Definim l'operació usant que els punts de la corba el·líptica sumen zero, si només si, estan alineats (i.e. hi ha una recta que els conté). Una recta sempre talla la corba el·líptica en 3 punts, comptant multiplicitats (Teorema de Bézout).

L'única recta que talla la corba en un sol punt de multiplicitat 3 és $s := \{Z = 0\}$. Llavors, l'element neutre és P_∞ per ser l'únic punt de la corba contingut en s. Per tant, sumar zero vol dir que el resultat obtingut és el neutre de l'operació: P_∞.

A partir d'això podem definir l'invers d'un punt P, que notem com $-P$, tal que serà el punt de tall de la la corba el·líptica amb la recta que passa per P i P_∞. Això es dedueix del fet que $P + (-P) + P_\infty$ sumen zero per estar alineats. Les rectes diferents a la de l'infinit que passen per $P_\infty = [0 : 1 : 0]$ tenen $X = c$ constant. Com que la recta passa per $P = (x, y)$ tenim que $X = x$. Substituint a l'equació de Weirestrass que defineix la corba obtenim que:

$$Y^2 + a_1xY + a_3Y = x^3 + a_2x^2 + a_4x + a_6.$$

Per tant, una solució és la que en dóna el punt P amb $Y = y$ i com que la suma de les Y ha de ser $-a_1x - a_3$ (deduït de restar a l'equació prèvia substituint les dues solucions, amb la mateixa x) l'altra solució serà $Y = -y - a_1x - a_3$. Així doncs:

$$-P = (x', y') = \begin{cases} x' = x, \\ y' = -y - a_1x - a_3. \end{cases} \tag{6}$$

Un cop definit l'element invers podem definir com sumem dos elements de la corba. Per fer-ho ho separem en tres casos diferents:

Si volem sumar el punt de l'infinit P_∞ a un altre punt P obtenim per definició: $P + P_\infty = P$.

Llavors, en cas de voler sumar dos punts de l'espai afí $P = (x_1, y_1),\ Q = (x_2, y_2)$ procedim de la següent manera:

Si els punts P i Q són inversos un de l'altre llavors obtenim que $P + Q = P_\infty$. Aquest és el cas si es compleixen les equacions deduïdes en (6), és a dir, si:

$$x_1 = x_2, \quad y_2 + y_1 + a_1x_1 + a_3 = 0.$$

Si aquest no és el cas, aleshores la recta r que passa per P i Q talla la corba en un altre punt afí $R = (x_3, y_3)$. Cal distingir entre dos casos, el cas en que $P \neq Q$ on $x_1 \neq x_2$ i el cas $P = Q$ on la recta r és la recta tangent a P. La recta tangent a la corba $f(x, y) = 0$ en el punt $P = (a, b)$ és:

$$\frac{\partial f}{\partial x}(P) \cdot (x - a) + \frac{\partial f}{\partial y}(P) \cdot (y - b) = 0.$$

Llavors, definim la recta r com $Y = \lambda X + \nu$ amb paràmetres:

$$\nu = y_1 - \lambda x_1, \quad \lambda = \begin{cases} \dfrac{y_2 - y_1}{x_2 - x_1} & x_1 \neq x_2, \\ \dfrac{3x_1^2 + a_2 x_1 + a_4 - a_1 y_1}{2y_1 + a_1 x_1 + a_3} & x_1 = x_2. \end{cases}$$

Substituint l'equació a la corba:

$$(\lambda X + \nu)^2 + a_1 X(\lambda X + \nu) + a_3(\lambda X + \nu) = X^3 + a_2 X^2 + a_4 X + a_6.$$

Com que $x_1,\ x_2,\ x_3$ són les arrels del polinomi, el podem expressar com:

$$(X - x_1)(X - x_2)(X - x_3) = 0.$$

Llavors, comparant els coeficients de grau dos dels dos polinomis s'obtenen les coordenades del tercer punt de tall:

$$x_3 = \lambda^2 + a_1\lambda - a_2 - x_1 - x_2, \quad y_3 = \lambda x_3 + \nu.$$

Com que $P + Q + R = P_\infty$ llavors $P + Q = -R$. Buscant l'invers del punt R calculat s'obté:

$$P + Q = (x', y') = \begin{cases} x' = \lambda^2 + a_1\lambda - a_2 - x_1 - x_2, \\ y' = -(\lambda + a_1)x' - \nu - a_3. \end{cases}$$

El següent pas és escriure els elements de la corba el·líptica en binari. La millor manera per la posterior execució de l'algorisme és representar els elements de la corba guardant el parell (x, y). Llavors, només cal escriure en binari les coordenades $x, y \in \mathbb{F}_q$.

Suposem que el cardinal del cos és diferent de 2 i 3. Una altre forma de procedir consisteix en escriure la corba (prenent arrels a (4)) com:

$$Y = \pm\sqrt{X^3 + aX + b} \qquad a, b \in \mathbb{F}_q.$$

Fent això podem representar els elements de la corba com (x, bit) on el bit extra ens diu si estem en una solució o l'altra de la y. Aquesta representació ocupa poca memòria, però cada cop que s'ha de fer servir y cal buscar el seu valor usant les dades, o sigui calculant les arrels quadrades de $X^3 + aX + b$.

Usar el grup $G = E(\mathbb{F}_p)$, amb p primer, per crear protocols criptogràfics ofereix una major seguretat que els grups multiplicatius. El principal avantatge del logaritme discret sobre corbes el·líptiques és que no es coneixen algorismes subexponencials per solucionar-lo, ja que no es pot aplicar l'algorisme del càlcul d'índex per no conèixer una base de factors d'aquest grup. Per tant, els millors algorismes que hi ha per trobar el logaritme discret en corbes el·líptiques sobre cossos $\mathbb{F}_p$ són els algorismes sobre grups generals.

El grup de les corbes el·líptiques també ofereix millores en lo referent a la implementació de l'algorisme en ordinadors. Usant corbes el·líptiques, podem canviar la corba $E(K)$ que fem servir sense canviar el cos $K = \mathbb{F}_q$ cosa que dóna més flexibilitat i opcions per triar el grup. Per fer-ho només cal canviar els paràmetres de la corba, ja que hi ha corbes amb nombres de punts diversos dins de l'interval de Hasse $[q + 1 - 2\sqrt{q}, q + 1 - 2\sqrt{q}]$. Al no canviar el cos K, les operacions de la corba funcionen de la mateixa manera. Per aquest motiu podem canviar de grup periòdicament sense canviar el software usat.

Capítol 2
Algorismes del logaritme discret en grups generals

Donats un grup cíclic G, un generador g i $h \in G$ volem calcular $\log_g(h)$. Per obtenir el logaritme discret existeixen diferents algorismes que es descriuen amb detall a continuació.

Observació 2.1. Segons el resultat que es pot trobar al teorema 1 de [**7**] existeix una fita inferior per qualsevol algorisme sobre grups generals. Sigui A un algorisme que només interactua amb els elements de G fent l'operació del grup i calculant inversos, sense fer servir l'estructura particular del grup. El cost de l'algorisme A és $\Omega(\sqrt{n})$. Aquest tipus d'algorismes s'anomenen mètodes de l'arrel quadrada (square root methods).

Tot i que treballem en grups generals, conèixer certes propietats de G o h ens permet reduir el cost de l'algorisme. Veurem algorismes que aprofiten que G és llis (definició 2.6 pàg. 36) per reduir el cost. N'hi ha altres on és útil saber que el logaritme discret de h està en un interval de mida w. Per exemple, aquest és el cas de l'algorisme de Kangaroo. També es poden fer modificacions en els algorismes de Shanks i força bruta per aconseguir el mateix resultat.

1. Algorismes de Força bruta

El logaritme discret és una aplicació entre grups finits. Per tant, provant tots els possibles exponents acabarem trobant el logaritme buscat. En aquest apartat veurem els algorismes més trivials per solucionar el logaritme discret, que es basen en aquesta idea.

Convé procedir de forma diferent segons si es vol calcular un o molts logaritmes. En el segon cas, és útil fer precomputacions per reduir significativament el cost de calcular cada logaritme.

Algorisme de força bruta

Per aquest algorisme es prenen $k \in \{0, 1, 2, ..., n-1\}$, començant per $k = 0$ i augmentant a cada pas. Llavors, es calcula g^k i es compara amb h. Si els resultats no coincideixen, s'augmenta k en una unitat i es repeteix el procés. El mètode s'aplica successivament fins a trobar la coincidència ($g^k = h$). Llavors $k = \log_g(h)$. (Per més detall veure l'algorisme B.2 de l'apèndix 2).

EXEMPLE 2.1. Prenem el grup $G = (\mathbb{Z}/p\mathbb{Z})^*$ amb $p = 19$.

Sabem que $|G| = 18$ i que té com a generadors $\{2, 3, 10, 13, 14, 15\}$. Prenem $g = 3$, recordem que per la propietat 1.2.3 sabem com canviar de base.

Llavors per calcular $\log_3(4)$ procedim de la manera següent:

$3^0 = 1,\ 3^1 = 3,\ 3^2 = 9,\ 3^3 = 8,\ 3^4 = 5,\ 3^5 = 15,\ 3^6 = 7,\ 3^7 = 2,\ 3^8 = 6,\ 3^9 = 18,\ 3^{10} = 16,\ 3^{11} = 10,\ 3^{12} = 11,\ 3^{13} = 14,\ 3^{14} = 4$. Per tant, $\log_3(4) = 14$.

Cost de l'algorisme

PROPOSICIÓ 2.1. *El cost de l'algorsime de força bruta és* $\text{cost}(n) = O(n)$. *Tant en el cas mitjà com en cas pitjor.*

DEMOSTRACIÓ.

Definim $X = \#\{\text{operacions del algorisme parant en el pas } s\} = s$

Definim l'exponent de h com $U \in \{0, 2, 3, ..., n-1\}$, que és una variable aleatòria (v.a.) uniforme.
Per tant,

$$E(X) = \sum_{s=1}^{n} s \cdot \underbrace{Pr(X = s)}_{U \text{ v.a. uniforme}} = \sum_{s=1}^{n} s \cdot \frac{1}{n} = \frac{1}{n} \sum_{s=1}^{n} s = \frac{1}{n} \frac{n(n-1)}{2} = \frac{n-1}{2} = O(n).$$

En el cas pitjor arribem fins a $k = n$. Per tant, $\text{cost}(n) = \sum_{s=1}^{n} O(1) = O(n)$. □

Algorisme de força bruta per múltiples logaritmes

Suposem que existeix un ordre total en el grup G. L'algorisme calcula el logaritme discret en dos passos, la precomputació i el càlcul explícit del logaritme. En la precomputació es calculen totes les exponencials del grup (i.e. g^k per $k \in \{0, 1, 2, ..., n-1\}$) i s'ordenen per la primera component els parells (g^k, k). A partir de la taula ordenada, només cal buscar cada logaritme ($\log_g(h)$) en temps lineal, usant cerca dicotòmica. (Per més detall veure l'algorisme B.3 de l'apèndix 2).

EXEMPLE 2.2. Prenem el mateix grup $G = (\mathbb{Z}/19\mathbb{Z})^*$ usat en l'exemple 2.1 i apliquem l'algorisme per calcular: $\log_3(2),\ \log_3(4),\ \log_3(6),\ \log_3(17)$.

El primer pas és fer la precomputació:

Calculem les exponencials: $A = \{(3^k, k)\} = \{(1,0),\ (3,1),\ (9,2),\ (8,3),\ (5,4),\ (15,5),\ (7,6),\ (2,7),\ (6,8),\ (18,9),\ (16,10),\ (10,11),\ (11,12),\ (14,13),\ (4,14),\ (12,15),\ (17,16),\ (13,17)\}$.

Ordenem A per la primera component: sort(A) $= \{(1,0),\ (2,7),\ (3,1),\ (4,14),\ (5,4),\ (6,8),\ (7,6),\ (8,3),\ (9,2),\ (10,11),\ (11,12),\ (12,15),\ (13,17),\ (14,13),\ (15,5),\ (16,10),\ (17,16)\}$

Llavors, podem calcular els logaritmes de manera senzilla (temps lineal) fent cerca dicotòmica (algorisme B.4) en la taula ordenada. Obtenim: $\log_3(2) = 7,\ \log_3(4) = 14,\ \log_3(6) = 8,\ \log_3(17) = 16$.

Cost de l'algorisme

PROPOSICIÓ 2.2. *El cost de la precomputació per al càlcul de múltiples logaritmes amb força bruta és* $cost(n) = O(n\log(n))$. *I calcular un logaritme un cop feta la precomputació té* $cost(n) = O(\log(n))$. *A més, el cost en espai és* $\text{cost}_E(n) = O(n)$.

DEMOSTRACIÓ.

Per calcular el cost de la precomputació observem que és el mateix que aplicar l'algorisme B.2 en cas pitjor. Llavors, per la prop. 2.1 tenim que el cost del bucle és $cost(n) = O(n)$.
Se sap que ordenar un vector A té $cost(n) = O(n\log(n))$, amb algorismes com per exemple mergesort.
Per tant, el cost de la precomputació és $O(n\log(n))$.

Per veure que l'algorisme de cerca dicotòmica té $cost(n) = O(\log(n))$ cal arribar a la recurrència següent:

$$C(n) = C(\lfloor n/2 \rfloor) + O(1).$$

La recurrència és clara, ja que en cada pas de l'algorisme reduïm a la meitat la mida del vector sobre el qual fem cerca dicotòmica.
Un cop tenim aquesta recurrència, veiem que es tracta de la mateixa recurrència que a la prop. 1.1 on es demostra que el cost és $C(n) = O(\log(n))$.

Per determinar el cost en espai s'observa que guardem una taula amb un total de n exponencials, on cadascuna ocupa un espai constant. Per tant, el cost és $\text{cost}_E(n) = O(n)$. □

2. Algorsime de Shanks (Baby-step Giant-step)

L'algorisme de Shanks aporta una millora en el cost respecte dels algorismes de força bruta. L'algorisme es basa en el teorema 2.3 que s'exposa a continuació.

TEOREMA 2.3. *Sigui n un enter positiu, $s = \lfloor\sqrt{n}\rfloor$ i $0 \leqslant k < n$. Podem escriure k com:*

$$k = q \cdot s + r, \quad amb\ 0 \leqslant r < s,\ 0 \leqslant q \leqslant s+1.$$

DEMOSTRACIÓ.

Usem l'algorisme de la divisió entera de k entre s: $k = q \cdot s + r$ on q és el quocient i r és el residu. Sabem per definició que el residu satisfà $0 \leqslant r < s$. Per arribar a la fita de q procedim de la següent manera:

$$q = \frac{k-r}{s} \leqslant \frac{k}{s} \leqslant \frac{n-1}{s} \leqslant \frac{n-1}{\sqrt{n}-1} = \sqrt{n}+1.$$

Com que q és un nombre natural tenim que $q \leqslant \lfloor\sqrt{n}+1\rfloor = \lfloor\sqrt{n}\rfloor = s+1$. □

Llavors, donat $h = g^k$ escrivim $k = sq + r$ d'on obtenim que $h = g^k = g^{sq+r} = g^r \cdot g^{sq}$. Transformant la igualtat anterior obtenim:

$$h \cdot g^{-sq} = g^r. \tag{7}$$

El teorema 2.3 ens garanteix que trobarem una igualtat (match) en l'equació (7) amb $0 \leqslant r < s$ i amb $0 \leqslant q \leqslant s+1$.

L'algorisme de Shanks consta de dues parts. La primera computa el Baby-step, on es fan passos en r que augmenta d'un en un. En aquesta part, es calculen els parells (g^r, r) i s'ordenen per la primera component en la taula S. La segona s'encarrega de computar el Giant-step fins a trobar el match, on es fan passos en $-sq$ que augmenta de s en s. En aquesta part, es calcula $h \cdot g^{-sq}$ i es busca el valor en S (cerca dicotòmica). Si no es troba en la taula, s'augmenta q en una unitat i es repeteix el procés. Un cop obtinguts r i q el logaritme discret és $\log_g(h) = sq + r$. (Per més detall veure l'algorisme B.5 de l'apèndix 2).

EXEMPLE 2.3. Prenem el mateix grup $G = (\mathbb{Z}/19\mathbb{Z})^*$ usat en exemples previs. Apliquem l'algorisme de Shanks per calcular $\log_3(4)$. Sigui $s = \lfloor\sqrt{19}\rfloor = 4$.

Computem el Baby-step $S = \{(3^r, r)\} = \{(1,0),\ (3,1),\ (9,2),\ (8,3)\}$.
$\text{Sort}(S) = \{(1,0),\ (3,1),\ (8,3),\ (9,2)\}$.

Busquem el match mentre computem el Giant-step $(4 \cdot 3^{-4q}, q)$. Per fer-ho sabem que $3^{-4} = 3^{14} = 4$. Llavors, a cada augment de q multipliquem per $3^{-4} = 4$ el

valor del pas anterior.
$(4,0)$ no match, $(4 \cdot 4, 1) = (16,1)$ no match, $(16 \cdot 4, 2) = (7,2)$ no match, $(7 \cdot 4, 3) = (9,3)$ match amb $(9,2)$ de la taula S.

Per tant, $q = 3,\ r = 2 \ \Rightarrow\ k = sq + r = 4 \cdot 3 + 2 = 14$. Així doncs, $\log_3(4) = 14$.

Cost de l'algorisme

PROPOSICIÓ 2.4. *El cost en temps de L'algorisme de Shanks de càlcul del logaritme discret és* $\text{cost}(n) = O(\sqrt{n}\log(n))$. *Tant en el cas pitjor com en el cas mitjà. L'algorisme ocupa un espai de* $\text{cost}_E(n) = O(\sqrt{n})$.

DEMOSTRACIÓ.

Per demostrar quin és el cost de l'algorisme es mira el cost de cadascun dels passos que es fan.

En el Baby-step es calcula g^r per $r \in \{0, ..., s-1\}$, es guarden aquests valors en S i s'ordena el conjunt. Computar totes les exponencials de S té un $\text{cost}(n) = \sum_{r=0}^{s-1} \theta(1) = \theta(s)$. L'ordenació té un cost de $O(s\log(s))$ aplicant un algorisme d'ordenació eficient, com és el cas de mergesort.

L'algorisme sempre executa tot el Baby-step. Per tant, si veiem que el cost del Giant-step és menor que el cost del Baby-step llavors el cost asimptòtic global serà el del primer pas.

En el Giant-step hem de computar cadascun dels $h \cdot g^{-sq}$ i per cadascun d'ells buscar el match en S amb cerca dicotòmica. Computar $h \cdot g^{-sq}$ té $\text{cost}(n) = \log(sq) \leqslant \log(s^2) = O(\log(s))$. Buscar el match té cost, com ja hem mencionat, $O(\log(s))$. Per tant, com que en el cas pitjor farem $s+1$ passos, obtenim que el cost del Giant-step és $(s+1)O(\log(s)) = O(s\log(s))$.

Així doncs, hem vist que l'algorisme té un $cost_{Total} = O(s\log(s)) = O(\sqrt{n}\log(n))$ que és el cost del Baby-step.

Per trobar el cost en espai només cal adonar-se que creem una taula amb s elements ($2s$ si considerem que cada parell (g^r, r) ocupa 2 posicions de memoria). A més d'aquesta taula, només cal guardar $(h \cdot g^{-sq}, q)$ que ocupa 2 posicions de memòria extres. Per tant, $\text{cost}(n) = O(s) = O(\sqrt{n})$. □

Observem que la quantitat d'espai que necessita aquest algorisme és $O(e^{\frac{1}{2}\log n})$, exponencial en els bits de n. Per tant, necessitem molta capacitat de memòria per executar l'algorisme de Shanks i, en canvi, la millora en temps d'execució només ens permet passar a un cost de $\sqrt{n}$ que segueix sent exponencial.

Nota: Si hem de computar una quantitat elevada de logaritmes sobre la mateixa base no cal repetir el Baby-step, ja que aquesta part de l'algorisme és la mateixa independentment del valor de h.

3. Algorismes Probabilístics

Un algorisme probabilístic es caracteritza per fer tries aleatòries durant l'execució de l'algorisme. Per començar, veiem-ne un exemple.

EXEMPLE 2.4. Sigui G un grup cíclic. L'algorisme per trobar un generador del grup G és un algorisme probabilístic. Es pren $g \in G$ aleatori i es comprova si és un generador. En cas que no ho sigui, es repeteix el procés fins a trobar-ne un.

Sigui n l'ordre del grup que descompon com $n = p_1^{\alpha_1} \cdot ... \cdot p_r^{\alpha_r}$. Observem que:

$$g \text{ és un generador de } G \Leftrightarrow g^{n/p_i} \neq 1 \quad \forall\, i \in \{1, ..., r\}. \tag{8}$$

Llavors, per comprovar si g és generador només cal calcular les r exponencials g^{n/p_i}.

Per veure la doble implicació de (8) observem el següent. Cap a la dreta, com que g és generador llavors $k = n$ és el primer enter tal que $g^k = 1$ i, per tant, es compleix la implicació. Cap a l'esquerra, suposem que $g^k = 1$ amb $k \mid n$. Llavors, $k = p_1^{\beta_1} \cdot ... \cdot p_r^{\beta_r}$ i si $\beta_i < \alpha_i$ tenim que $k \leqslant n/p_i$ d'on s'obté $g^{n/p_i} = 1$. Per tant, $k = n$ i g és generador.

Observem que el temps d'execució de l'algorisme depèn de les tries aleatòries de g. El cas mitjà té cost polinòmic, ja que comprovar si g és generador té cost polinòmic i de mitjana cal prendre un nombre polinòmic d'elements fins a trobar-ne un que generi G. Això últim es dedueix de $\Pr(g \text{ generador}) \geqslant \frac{\pi(n)}{n} = \frac{1}{\log n}$, on $\pi(n)$ és la quantitat de primers menors que n, que pel teorema dels nombres primers compleix $\pi(n) \sim \frac{n}{\log n}$.

Ens serà útil conèixer i diferenciar dos tipus d'algorismes probabilístics: els algorismes de las Vegas i els algorismes de Monte Carlo.

DEFINICIÓ 2.1. *Direm que tenim un algorisme de Las Vegas si l'algorisme sempre dóna resultats correctes del problema i l'única diferencia entre execucions és el temps fins a obtenir el resultat.*

Per exemple, l'algorisme per trobar un generador en un grup cíclic és un algorisme de Las Vegas. Hi ha dues variants de la definició d'un algorisme de Las Vegas, en una obtenir el resultat correcte vol dir que s'obté el resultat del problema i en l'altra s'accepta el cas en que l'algorisme informa que ha fallat. Per

passar del segon al primer cas, només cal executar l'algorisme repetidament fins que aquest dóna la solució del problema.

DEFINICIÓ 2.2. *Direm que tenim un algorisme de Monte Carlo si l'algorisme obté el resultat del problema, excepte en alguns casos amb baixa probabilitat. En aquests casos aïllats, l'algorisme pot equivocar-se o no donar cap resultat.*

Els algorismes de Monte Carlo no donen cap informació sobre si la sortida és un resultat correcte.

3.1. Algorisme ρ de Pollard.

L'algorisme ρ de Pollard és un algorisme probabilístic molt usat a la pràctica, ja que és un dels que ofereix millors resultats per calcular el logaritme discret sobre un grup qualsevol, en particular per corbes el·líptiques.

Aquest algorisme es basa en crear una seqüència pseudo-aleatòria d'elements de G (i.e. $W = \{x_0, x_1, x_2, ...\}$ amb $x_i \in G$) fins a trobar un cicle, també anomenat match o col·lisió. La seqüència creada W s'anomena camí pseudo-aleatori determinista (deterministic pseudo-random walk).

DEFINICIÓ 2.3. *Direm que hem trobat un match o col·lisió en una seqüència W d'elements de G sii $\exists\ i, j \geqslant 0$ amb $i < j$ tals que $x_i = x_j$.*

Per crear la seqüència W s'aplica iterativament una funció de Pollard f (i.e. $x_i = f(x_{i-1}) = f^i(x_0)$). Llavors, elegint $x_0 \in G$ la seqüència queda determinada. Aquí, es pren $x_0 = g$ encara que també es pot agafar $x_0 = g^r \cdot h^s$ per qualsevol $r, s < n$. És clar que la seqüència creada per l'algorisme sempre tindrà un match, ja que el grup G és finit i $x_i \in G\ \forall i \geqslant 0$.

Per definir una funció de Pollard se separa G en r subconjunts (no necessàriament subgrups) usant una funció:

$$S:\ G \longrightarrow R = \{1, ..., r\}, \tag{9}$$

on els subconjunts són $S_i = \{x \in G \mid S(x) = i\} = S^{-1}(i)\ \ \forall i \in R$.

En el document original de Pollard es fa servir la següent funció amb $r = 3$ per crear la seqüència W:

$$x_{i+1} = f_P(x_i)\ =\ \begin{cases} g \cdot x_i & x_i \in S_1, \\ h \cdot x_i & x_i \in S_2, \\ x_i^2 & x_i \in S_3. \end{cases}$$

En articles posteriors s'ha millorat la funció original usada per Pollard. S'agafa $M_j = g^{u_j} \cdot h^{v_j}$ per $j \in \{1, ..., r\}$ i $u_j, v_j \in \{0, ..., n-1\}$.

DEFINICIÓ 2.4. *Una funció de Pollard generalitzada es defineix com:*

$$x_{i+1} = f_{PG}(x_i) \;=\; \begin{cases} M_j \cdot x_i & per\ S(x_i) = j \neq r, \\ x_i^2 & per\ S(x_i) = r. \end{cases}$$

DEFINICIÓ 2.5. *Una funció de Pollard additiva es defineix com:*

$$x_{i+1} = f_{PA}(x_i) \;=\; M_j \cdot x_i \quad per\ S(x_i) = j.$$

Per tant, usant les funcions pseudo-aleatories definides obtenim que $x_i = g^{a_i} \cdot h^{b_i}$. On coneixem la seqüència per a les a's i les b's:

Funció de Pollard Generalitzada	Funció de Pollard Additiva
$a_{i+1} \;=\; \begin{cases} 2a_i & (mod\ n) \quad si\ S(x_i) = r \\ a_i + u_{S(x_i)} & (mod\ n) \quad si\ S(x_i) \neq r \end{cases}$	$a_{i+1} \;=\; a_i + u_{S(x_i)}\ (mod\ n)$
$b_{i+1} \;=\; \begin{cases} 2b_i & (mod\ n) \quad si\ S(x_i) = r \\ b_i + v_{S(x_i)} & (mod\ n) \quad si\ S(x_i) \neq r \end{cases}$	$b_{i+1} \;=\; b_i + v_{S(x_i)}\ (mod\ n)$

Definim la funció walk on es realitza tot el càlcul del camí pseudo-aleatori tal com s'ha explicat:

$$(x_{i+1}, a_{i+1}, b_{i+1}) \;=\; \text{walk}(x_i, a_i, b_i).$$

Un cop definida la funció necessitem un algorisme eficient que ens informi quan tinguem un match en la seqüència. L'algorisme usat per Pollard per trobar el match és l'algorisme de Floyd (Floyd's collision algorithm) que s'explica a continuació (Per més detall veure l'algorisme B.6 de l'apèndix 2).

L'algorisme de Floyd no compara el nou element x_i amb tots els anteriors (molt ineficient), sinó que només es compara x_{2i} amb l'element x_i. I així successivament, $x_{2(i+1)}$ es compara amb x_{i+1}, on haurem d'aplicar dos cops la funció walk.

Per fer servir l'algorisme de Floyd necessitem que la nostra seqüència W no sigui aleatòria, sinó que sigui una seqüència pseudo-aleatòria creada a partir de f. D'aquesta manera, donat el primer match de la seqüència $x_{i_0} = x_{j_0}$ $(i_0 < j_0)$ tenim que $x_{i_0+\alpha} = f^{\alpha}(x_{i_0}) = f^{\alpha}(x_{j_0}) = x_{j_0+\alpha}$. Així doncs, tenim un match a la seqüència quan:

$$x_u = x_v, \quad \text{on } u, v \geqslant i_0 \text{ tals que } u \equiv v \mod m, \text{ amb } m = j_0 - i_0.$$

Llavors, per $u = C \cdot m \Rightarrow x_u = x_{2u}$ ja que $2u - u = u = C \cdot m \equiv 0 \pmod{m}$. El primer múltiple de m per sobre de i_0 és $u = m \cdot \left\lceil \frac{i_0}{m} \right\rceil$, que es pot veure fàcilment que no supera j_0. Aquest és el valor que trobarà l'algorisme de Floyd.

Un cop trobat el match tenim que $x_i = x_j$ on $x_i = g^{a_i} \cdot h^{b_i}$ i $x_j = g^{a_j} \cdot h^{b_j}$. Per tant, obtenim la següent equació:

$$g^{a_i} \cdot h^{b_i} \;=\; g^{a_j} \cdot h^{b_j}.$$

Llavors, excepte en el cas degenerat (en que no existeix $(b_j - b_i)^{-1}$) obtenim:

$$h \;=\; g^{(a_i - a_j)(b_j - b_i)^{-1} \bmod n}.$$

OBSERVACIÓ 2.2. Si l'ordre del grup n és primer la condició que ens assegura que podem invertir $b_j - b_i$ és $b_i \equiv b_j \pmod{n}$. Si n no fos primer tindríem elements $b_j - b_i \not\equiv 0$ no invertibles mòdul n. Observem això a l'exemple 2.5.

EXEMPLE 2.5. Prenem $G = (\mathbb{Z}/19\mathbb{Z})^*$, on $n = 18$. Apliquem l'algorisme de ρ de Pollard per calcular $\log_3(4)$.

Definim la funció de Pollard $f_P(x_i) \;=\; \begin{cases} 3 \cdot x_i & x_i \in S_1 = \{1,2,3,4,5,6\}, \\ 4 \cdot x_i & x_i \in S_2 = \{7,8,9,10,11,12\}, \\ x_i^2 & x_i \in S_3 = \{13,14,15,16,17,18\}. \end{cases}$

Apliquem l'algorisme:

$x_0 = 3$	$a_0 = 1$	$b_0 = 0$			
$x_1 = f(3) = 9$	$a_1 = 2$	$b_1 = 0,$	$x_2 = f^2(3) = 17$	$a_2 = 2$	$b_2 = 1,$
$x_1 = f(9) = 17$	$a_1 = 2$	$b_1 = 1,$	$x_2 = f^2(17) = 12$	$a_2 = 5$	$b_2 = 2,$
$x_1 = f(17) = 4$	$a_1 = 4$	$b_1 = 2,$	$x_2 = f^2(12) = 2$	$a_2 = 5$	$b_2 = 4,$
$x_1 = f(4) = 12$	$a_1 = 5$	$b_1 = 2,$	$x_2 = f^2(2) = 18$	$a_2 = 7$	$b_2 = 4,$
$x_1 = f(12) = 10$	$a_1 = 5$	$b_1 = 3,$	$x_2 = f^2(18) = 3$	$a_2 = 15$	$b_2 = 8,$
$x_1 = f(10) = 2$	$a_1 = 5$	$b_1 = 4,$	$x_2 = f^2(3) = 17$	$a_2 = 16$	$b_2 = 9,$
$x_1 = f(2) = 6$	$a_1 = 6$	$b_1 = 4,$	$x_2 = f^2(17) = 12$	$a_2 = 33$	$b_2 = 18,$
$x_1 = f(6) = 18$	$a_1 = 7$	$b_1 = 4,$	$x_2 = f^2(12) = 2$	$a_2 = 33$	$b_2 = 20,$
$x_1 = f(18) = 1$	$a_1 = 14$	$b_1 = 8,$	$x_2 = f^2(2) = 18$	$a_2 = 35$	$b_2 = 20,$
$x_1 = f(1) = 3$	$a_1 = 15$	$b_1 = 8,$	$x_2 = f^2(18) = 3$	$a_2 = 71$	$b_2 = 40.$

Hem trobat un match en la seqüència. Per tant:

$$3^{15} \cdot 4^8 = 3^{71} \cdot 4^{40} \;\Rightarrow\; 4^{8-40} = 3^{71-15} \;\Rightarrow\; 4 = 3^{(71-15)(8-40)^{-1}} = 3^{2 \cdot 4^{-1}} = \nexists.$$

Això es deu a que $n = 18$ no és primer i 4 no és invertible per la multiplicació en $\mathbb{Z}/18\mathbb{Z}$ ja que $4 \cdot 9 = 36 \equiv 0 \; (mod \; 18)$. En aquest cas l'algorisme de Pollard falla.

Aquest fet ens porta a voler veure quina és la probabilitat que l'algorisme de Pollard no doni la solució del problema. Anomenem a aquest cas degenerat. Com s'afirma en [**8**], no hi ha una fita general per a la probabilitat d'estar en

el cas no degenerat. Tot i així, el teorema següent ens diu que trobarem una solució no degenerada amb alta probabilitat per a certs grups.

TEOREMA 2.5. *Considerem l'algorisme ρ de Pollard sobre un grup G d'ordre primer, on comencem en un punt aleatori $x_0 = g^{r_1} \cdot h^{r_2}$. Suposem que l'ordre de 2 (mod n) és major o igual que $c_0(\log n)^3$ amb c_0 constant adequada. Llavors, qualsevol col·lisió que es doni abans de temps T és no degenerada amb probabilitat com a mínim $1 - \frac{3}{2}\frac{T^2}{n^2}$. En particular, com l'algorisme té cost esperat en temps de $O(\sqrt{n})$ llavors la probabilitat de tenir una colisió no degenerada és almenys $1 - O(\frac{1}{n})$.*

DEMOSTRACIÓ. Veure el Teorema 1.2 de [**8**]

Encara que no fem la demostració d'aquest resultat, sembla raonable pensar que el cas degenerat ($b_1 \equiv b_2$) tingui una probabilitat de $O(\frac{1}{n})$. Si suposem que els elements de la seqüència es prenen de forma aleatòria i els exponents s'agafen mòdul n, hi ha n possibles valors de $b_1 - b_2$, per tant, seran congruents amb zero amb probabilitat $O(\frac{1}{n})$. □

Nota: L'algorisme de ρ de Pollard s'anomena així perquè si dibuixem la seqüència fins a la col·lisió de forma adequada aquesta té la forma d'una ρ. Donat el primer match ($x_i = x_j$) es dibuixa el camí aleatori amb una cua (que és la part de $x_1, ..., x_i$ no-cíclica) seguit d'un cicle (que és la part $x_{i+1}, ..., x_j$).

Cost de l'algorisme

TEOREMA 2.6 (Paradoxa de l'aniversari). *Sigui $\mathcal{S}$ un conjunt de N elements. Si prenem una seqüència d'elements amb distribució uniforme en $\mathcal{S}$ i cada element independent llavors el nombre esperat d'elements que cal agafar fins que hi hagi un element repetit és menor que $\sqrt{\frac{\pi N}{2}} + 1 \approx 1.253\sqrt{N}$.*

DEMOSTRACIÓ.

Definim $X = \#\{\text{elements seleccionats aleatòriament de } \mathcal{S} \text{ fins a tenir repetició}\}$

Un cop seleccionats ℓ elements la probabilitat que el següent sigui diferent que els anteriors és $(1 - \ell/N)$. Per tant:

$$Pr(X > \ell) = 1(1 - 1/N)(1 - 2/N) \cdot ... \cdot (1 - (\ell - 1)/N).$$

Llavors, fent servir la desigualtat $1 - x \leqslant e^{-x}$ obtenim que:

$$Pr(X > \ell) \leqslant 1e^{-1/N}e^{-2/N} \cdot ... \cdot e^{-(\ell-1)/N} = e^{\sum_{j=0}^{\ell-1} j/N} = e^{-\frac{\ell(\ell-1)}{2N}} \leqslant e^{-\frac{(\ell-1)^2}{2N}}.$$

Així doncs, si calculem l'esperança de X obtenim:

$E(X) = \sum_{\ell=1}^{\infty} \ell Pr(X = \ell) = \sum_{\ell=1}^{\infty} \ell(Pr(X > \ell - 1) - Pr(X > \ell)) = \sum_{z=0}^{\infty}(z + 1)Pr(X > z) - \sum_{\ell=1}^{\infty} \ell Pr(X > \ell) = \sum_{\ell=0}^{\infty}(\ell + 1 - \ell)Pr(X > \ell) = \sum_{\ell=0}^{\infty} Pr(X > \ell) \leqslant 1 + \sum_{\ell=1}^{\infty} e^{-(\ell-1)^2/2N}$.

Aproximant la sèrie anterior per una integral obtenim:

$$E(X) = 1 + \int_0^{\infty} e^{-x^2/2N} dx.$$

Fent el canvi de variables $u = x/\sqrt{2N}$ obtenim:

$$E(X) = 1 + \sqrt{2N} \int_0^{\infty} e^{-u^2} du = 1 + \sqrt{2N} \cdot \frac{\sqrt{\pi}}{2} = 1 + \sqrt{\frac{N\pi}{2}}.$$

□

El teorema 2.6 ens dóna el cost de l'algorisme de la rho de Pollard $\text{cost}(n) = O(\sqrt{n})$ per un grup G d'ordre n. S'està suposant que tenim una seqüència generada aleatòriament. Com sabem aquest no és el nostre cas, ja que la seqüència de l'algorisme es va creant iterant una funció f. Tot i així, si la funció f parteix d'un x_0 aleatori l'argument segueix sent vàlid. Cal tenir en compte que per obtenir el cost de l'algorisme es suposa que fer una iteració de f té cost constant (cert per les funcions que s'han definit).

L'algorisme de Pollard aporta una millora significativa en l'espai de memòria necessari, ja que només cal guardar dos triplets (x_i, a_i, b_i) de valors que anem actualitzant a cada pas.

3.2. Algorisme del cangur (Kangaroo algorithm).

L'algorisme del cangur és especialment útil per calcular logaritmes discrets en un interval més petit que la mida del grup n. Suposem h tal que $h = g^k$ on sabem que $a \leqslant k < a + w$. Llavors, la complexitat d'aquest algorisme depèn de w i no de l'ordre del grup n. A més, l'algorisme segueix funcionant en cas de no tenir cap informació de l'element del grup h, prenent $w = n$. Per simplicitat, es calcula $\widetilde{h} = h \cdot g^{-a}$. Llavors, $\widetilde{h} = g^{\widetilde{k}}$ amb $0 \leqslant \widetilde{k} < w$. Per tant, podem suposar $a = 0$ sense pèrdua de generalitat.

L'algorisme original va ser presentat per Pollard, però en aquest apartat expliquem l'algorisme millorat per Van Oorschot i Wiener usant punts distingits, punts que comparem per buscar el match. (Per més detall veure l'algorisme B.7 de l'apèndix 2).

Siguin $W_1 = \{x_0, x_1, x_2, ...\}$ i $W_2 = \{y_0, y_1, y_2, ...\}$ dues seqüències on $x_0 = g^{\lfloor w/2 \rfloor}$ i $y_0 = h$. Llavors, $x_{i+1} = f(x_i)$ i $y_{i+1} = f(y_i)$ on f és una funció definida de la forma següent:

$$f(z) \;=\; g^{u_j} \cdot z, \;\; per \; S(z) = j \qquad \text{on } 0 \leqslant u_j < \sqrt{w}, \;\; z \in G.$$

Es defineix S de la mateixa forma que en les funcions de Pollard (9) de l'apartat anterior. Observem que els salts que realitza la funció f són petits (no apliquem z^2 ja que fa un salt massa gran i restringim $u_j < \sqrt{w}$). A més, volem salts coneguts per aquest motiu no multipliquem per h, ja que no coneixem k tal que $h = g^k$. Per exemple, en aquest treball farem servir la funció següent:

$$f(z) \;=\; \begin{cases} g \cdot z & z \in S_1 = \{z \mid S(z) = 1\}, \\ g^2 \cdot z & z \in S_2 = \{z \mid S(z) = 2\}, \\ g^4 \cdot z & z \in S_3 = \{z \mid S(z) = 3\}. \end{cases} \tag{10}$$

La idea de l'algorisme, tal com explicava Pollard, tracta de "atrapar un cangur salvatge amb un cangur domesticat", on els salts del camí W $(x_{i-1} \;\rightarrow\; x_i)$ són els salts del cangur i els elements visitats (x_i) són les petjades.

El cangur domesticat és la seqüència $x_i = g^{a_i} = g^{\lfloor w/2 \rfloor} \cdot g^{\widetilde{a}_i}$ on coneixem cada $\widetilde{a}_i$ i partim de $\lfloor w/2 \rfloor$ conegut. El cangur salvatge és la seqüència $y_i = h \cdot g^{b_i} = g^k \cdot g^{b_i}$ on coneixem cada b_i però partim de k desconegut. Els cangurs van deixant trampes (punts distingits) fins que un d'ells cau a la trampa de l'altre, cas en que tenim la col·lisió buscada.

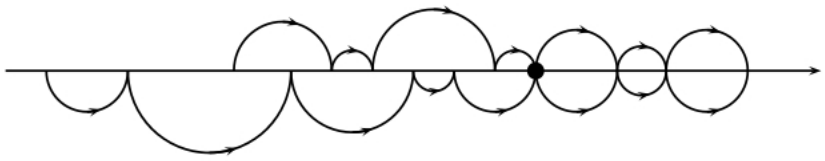

FIGURA 1. Representació del Kangaroo walk. El cangur domesticat dibuixat per sobre l'eix i el cangur salvatge per sota. Els salts dels cangurs representen els salts a l'exponent i el punt la primera col·lisió.

Procedim, doncs, de la forma esmentada per calcular els camins. Un pas de l'algorisme per calcular la seqüència serà:

$$(x_{i+1}, a_{i+1}) \;=\; \text{walk}(x_i, a_i), \;\; (y_{i+1}, b_{i+1}) \;=\; \text{walk}(y_i, b_i).$$

Cal adonar-se que no és necessari agafar molts punts distingits, ja que a partir del primer match $(x_i = y_j)$ tenim que $x_{i+\alpha} = x_{j+\alpha}$. Així doncs, si agafem un

punt distingit en x i un en y per sobre del match l'algorisme troba la col·lisió. Llavors, es compleix que:

$$x_i = g^{a_i} = h \cdot g^{b_j} = y_j \;\Rightarrow\; h = g^{a_i - b_j}.$$

Nota: El mètode també rep el nom d'algorisme λ de Pollard, ja que si dibuixem les seqüències de forma adequada aquestes formen d'una λ. Donat el primer match ($x_i = y_j$) es dibuixen les seqüències en diagonal i en sentit ascendent, formant les potes de la λ. Al trobar-se en x_i i y_j continuen juntes ($x_{i+1} = y_{j+1}, ..., x_{i+r} = y_{j+r}$).

EXEMPLE 2.6. Donat $G = (\mathbb{Z}/19\mathbb{Z})^*$, on $n = 18 = w$. Volem calcular $\log_3(4)$ amb l'algorisme de Kangaroo.

Farem servir la funció (10) amb $g = 3$ per crear les seqüències generades pels cangurs. Definim els conjunts que separen G com:

$$S_1 = \{1,2,3,4,5,6\}, \quad S_1 = \{7,8,9,10,11,12\}, \quad S_1 = \{13,14,15,16,17,18\}.$$

Definim el conjunt de punts distingits $D = \{x \mid x \equiv 0 \pmod 3\}$ i requadrarem els punts distingits per diferenciar-los de la resta. Un cop definits els elements que intervenen en l'algorisme, l'apliquem per trobar el logaritme discret:

$$\begin{cases} x_0 = 3^9 = \boxed{18} & a_0 = 9, \\ y_0 = 4 & b_0 = 0. \end{cases}$$

$$\begin{cases} x_1 = 3^4 \cdot 18 = 14 & a_1 = 13, \\ y_1 = 3 \cdot 4 = \boxed{12} & b_1 = 1. \end{cases}$$

$$\begin{cases} x_2 = 3^4 \cdot 14 = 13 & a_2 = 17, \\ y_2 = 3^2 \cdot 12 = 13 & b_2 = 3. \end{cases}$$

$$\begin{cases} x_3 = 3^4 \cdot 13 = 8 & a_3 = 21, \\ y_3 = 3^4 \cdot 13 = 8 & b_3 = 7. \end{cases}$$

$$\begin{cases} x_4 = 3^2 \cdot 8 = \boxed{15} & a_4 = 23, \\ y_4 = 3^2 \cdot 8 = \boxed{15} & b_4 = 9. \end{cases}$$

Per tant, obtenim que $k = \log_3(4) = 23 - 9 = 14$.

Cost de l'algorisme

TEOREMA 2.7. *Donat D el conjunt de punts distingits. Suposem que* $\Pr(x \in D) = \frac{c \log w}{\sqrt{w}}$ *i que* $m \approx \sqrt{w}/2$ *és el salt mitjà dels cangurs. Llavors, l'algorisme del cangur té un cost esperat en temps de* $(2 + o(1))\sqrt{w}$. *A més, el cost esperat en espai de l'algorisme serà* $O(\log w)$.

Demostració.

Ordenem els elements de G pel seu exponent, on no sabem en quina posició es troba h. Llavors, per minimitzar la distancia entre $y_0 = h$ i $x_0 = g^r$ prenem $r = \lfloor w/2 \rfloor$. D'aquesta manera, la distància mitjana entre y_0 i x_0 és $w/4$. Per tant, el nombre mitjà de salts que ha de fer el cangur de darrere per arribar a la posició inicial del cangur de davant és $w/4m$.

A partir d'aquest punt el cangur de darrere estarà en una regió que ha estat trepitjada pel cangur de davant. Per tant, suposant que tenim una distribució uniforme de petjades, per cada regió de mida m de mitjana hi haurà una petjada del cangur davanter. Així doncs, la probabilitat que en un salt el cangur de darrere no vagi a parar en una petjada del cangur de davant és $1 - 1/m$. Llavors, el nombre de salts esperat fins a tenir una coincidència és m (ja que $1 = \#\text{Coincidències} = \#\text{Salts} \cdot \frac{1}{m}$).

Ens falta afegir el nombre de salts fins que el cangur de darrere troba un punt distingit, un cop els dos cangurs ja han coincidit en un punt (no distingit). El nombre esperat de salts serà $\frac{1}{\Pr(x \in D)} = \frac{1}{c \log w/\sqrt{w}} = \frac{\sqrt{w}}{c \log w}$ (ja que $1 = \#\text{Punts Distingits} = \#\text{Salts} \cdot \Pr(x \in D)$).

Ajuntant tots els passos, prenent $m = \sqrt{w}/2$ òptim i tenint en compte que hi ha 2 cangurs obtenim el següent cost:

$$2 \cdot \left(\frac{w}{4m} + m + \frac{\sqrt{w}}{c \log w} \right) = 2 \cdot \left(\frac{\sqrt{w}}{2} + \frac{\sqrt{w}}{2} + \frac{\sqrt{w}}{c \log w} \right) = (2 + o(1))\sqrt{w}.$$

A més, com el nombre esperat de passos que fa l'algorisme és $O(\sqrt{w})$ i $\Pr(x \in D) = c \log w/\sqrt{w}$ llavors en total guardem en memòria una quantitat de punts distingits de $O(\sqrt{w}) \cdot \dfrac{c \log w}{\sqrt{w}} = O(\log w)$. □

Cal fer notar que l'algorisme no sempre convergeix. Hi ha la possibilitat que els cangurs no es trobin mai, en cas de tenir cicles disjunts. Tot i així, triant la funció walk de la forma explicada hi ha poques possibilitats de que això passi.

4. Algorismes per calcular el logaritme discret sobre grups llisos

En aquesta secció, s'estudia el problema del logaritme discret en grups G que tenen la propietat de ser llisos. Aquesta propietat permet reduir el cost de solucionar el logaritme discret.

DEFINICIÓ 2.6. *Direm que un nombre n és llis (smooth) si compleix que factoritza com a producte de primers petits. Més precisament, direm que n és B-smooth o B-llis si factoritza com a producte de primers menors que B:*

$$n = p_1^{\alpha_1} \cdot \ldots \cdot p_r^{\alpha_r} \quad amb\ p_i \leqslant B.$$

Donat un grup G d'ordre n direm que és B-llis si el seu ordre ho és.

Sigui G un grup llis. Veiem la forma de reduir el cost de calcular el logaritme discret sobre aquest grup en els teoremes següents.

TEOREMA 2.8. *Sigui $G = \langle g \rangle$ un grup tal que $|G| = n_1 \cdot n_2$, on $\mathrm{mcd}(n_1,\ n_2) = 1$. Prenem $G_1 = \langle g^{n_2} \rangle$ i $G_2 = \langle g^{n_1} \rangle$. Es té que $|G_1| = n_1$ i $|G_2| = n_2$.*

Llavors, l'aplicació següent és un isomorfisme:

$$\begin{array}{ccc} G & \longleftrightarrow & G_1 \times G_2 \\ x & \longrightarrow & f(x) = (x^{n_2}, x^{n_1}) \\ f^{-1}(y,z) = y^a \cdot z^b & \longleftarrow & (y,z) \end{array}$$

On es compleix que $an_2 + bn_1 = 1$.

DEMOSTRACIÓ.

$\exists\ a, b$ tals que $an_2 + bn_1 = 1$, ja que n_1 i n_2 són coprimers i la igualtat anterior és la identitat de Bézout.

$f(x)$ està ben definida ja que $x^{n_2} \in G_1 = \langle g^{n_2} \rangle$ pel fet de que $x^{n_2} = (g^k)^{n_2} = (g^{n_2})^k$. Equivalentment, $x^{n_1} \in G_2$. També $f^{-1}(y,z)$ està ben definida ja que $f^{-1}(y,z) = g^k \in G = \langle g \rangle$ per un cert k.

Llavors, falta per veure que són inverses una de l'altra.

$$(f \circ f^{-1})(y,z) = f(y^a \cdot z^b) = ((y^a \cdot z^b)^{n_2},\ (y^a \cdot z^b)^{n_1}) = (y,z).$$

Per comprovar l'última igualtat veiem que:
(1) $y^{an_2} \cdot z^{bn_2} = y^{an_2} \cdot 1 = y^{1-bn_1} = y$ on hem usat que si $x \in G_i \Rightarrow x^{n_i} = 1$.
(2) $y^{an_1} \cdot z^{bn_1} = 1 \cdot z^{bn_1} = z^{1-an_2} = z$ on hem usat que si $x \in G_i \Rightarrow x^{n_i} = 1$.

Per l'altra composició de funcions, també obtenim la identitat:

$$(f^{-1} \circ f)(x) = f^{-1}(x^{n_2}, x^{n_1}) = x^{an_2} \cdot x^{bn_1} = x.$$

□

Suposant coneguts els logaritmes $\log_{g^{n_2}}(h^{n_2}) = k_2,\ \log_{g^{n_1}}(h^{n_1}) = k_1$ es té que:

$$\log_g(h) = k_2 \cdot n_2 \cdot a + k_1 \cdot n_1 \cdot b.$$

Això es comprova veient el següent:

$$h = h^{an_2+bn_1} = (h^{n_2})^a \cdot (h^{n_1})^b = ((g^{n_2})^{k_2})^a \cdot ((g^{n_1})^{k_1})^b = g^{k_2 n_2 a + k_1 n_1 b}.$$

El procés es pot iterar repetidament si al seu torn $n_i = \widetilde{n}_1 \cdot \widetilde{n}_2$. En particular, sent $|G| = p_1^{\alpha_1} \cdot ... \cdot p_r^{\alpha_r}$ podem calcular el logaritme discret sobre grups d'ordre $|G_i| = p_i^{\alpha_i}$.

El teorema 2.8 ens diu la manera de reduir el càlcul logaritme discret en un grup G a grups de cardinal potència de primer. Considerem ara $|G| = p^\alpha$ amb p primer. El teorema següent ens permet reduir el problema de calcular el logaritme discret en G a fer-ho sobre grups de cardinal p.

TEOREMA 2.9. *Donat G tal que $|G| = p^\alpha$. Sigui $\widetilde{G} = \langle g^{p^{\alpha-1}} \rangle$ amb $|\widetilde{G}| = p$. Llavors, es pot calcular $\log_g(x) = k$ resolent α logaritmes sobre l'aplicació:*

$$\log_{g^{p^{\alpha-1}}} : \widetilde{G} \longrightarrow \mathbb{Z}/p\mathbb{Z}$$

DEMOSTRACIÓ.

Sabem que $\log_g(x) = k$ compleix $0 \leqslant k < p^\alpha$. Per tant, escrivint k en base p:

$$k = c_0 + c_1 p + c_2 p^2 + ... + c_{\alpha-1} p^{\alpha-1} \quad \text{amb } 0 \leqslant c_i < p \ \ \forall i \in \{0, 1, ..., \alpha - 1\}.$$

Per trobar $c_0, c_1, ...$ procedim de forma seqüencial:

Per calcular c_0 fem:

$$x^{p^{\alpha-1}} = (g^k)^{p^{\alpha-1}} = g^{(c_0+c_1 p+c_2 p^2+...)p^{\alpha-1}} = g^{c_0 p^{\alpha-1}} \cdot \underbrace{g^{(c_1+c_2 p+...)p^\alpha}}_{1} = (g^{p^{\alpha-1}})^{c_0}.$$

Per calcular c_j amb $j > 0$ fem:

$$x^{p^{\alpha-j-1}} = (g^k)^{p^{\alpha-j-1}} = g^{kp^{\alpha-j-1}} = g^{(c_0+c_1 p+...+c_j p^j+...+c_{\alpha-1}p^{\alpha-1})p^{\alpha-j-1}} \Rightarrow$$

$$\Rightarrow \ x^{p^{\alpha-j-1}} = g^{(c_0+c_1 p+...+c_{j-1}p^{j-1})p^{\alpha-j-1}} \cdot g^{(c_j p^j)p^{\alpha-j-1}} \cdot \underbrace{g^{(c_{j+1}+c_{j+2}p+...)p^\alpha}}_{1},$$

on $c_0, ..., c_{j-1}$ són coneguts. Llavors, obtenim que:

$$(x \cdot g^{-(c_0+c_1 p+...+c_{j-1}p^{j-1})})^{p^{\alpha-j-1}} = (g^{p^{\alpha-1}})^{c_j}.$$

□

Veiem un exemple en que s'apliquen els teoremes 2.8 i 2.9 per reduir el cost del logaritme discret.

EXEMPLE 2.7. Donat $G = (\mathbb{Z}/13\mathbb{Z})^*$, on $|G| = 12 = 2^2 \cdot 3$. Volem calcular el $\log_2(7)$ fent servir el mètode explicat (Teorema 2.8).

El primer pas consisteix en solucionar l'equació diofàntica: $4a + 3b = 1$ on $a = 1,\ b = -1$ compleixen l'equació.

Llavors, prenem els subgrups de G següents $G_1 = \langle 2^4 \rangle,\ G_2 = \langle 2^3 \rangle$ amb cardinals $|G_1| = 3,\ |G_2| = 4$.

Calculem els logaritmes corresponents $\begin{cases} \log_{2^4}(7^4) = \log_3(9) = k_1 = 2, \\ \log_{2^3}(7^3) = \log_8(5) = k_2 = 3. \end{cases}$

Així doncs, obtenim que:

$$\log_2(7) = 4 \cdot a \cdot k_1 + 3 \cdot b \cdot k_2 = 4 \cdot 2 \cdot 1 + 3 \cdot 3 \cdot (-1) = 8 - 9 = 11 (\text{mod } 12).$$

Per al calcul de $\log_8(5)$ podem procedir com indica el Teorema 2.9. Considerem:

$$k = \log_8(5) = c_0 + c_1 \cdot 2.$$

Sigui $\overline{G} = \langle (2^3)^2 \rangle = \langle 12 \rangle$. D'on calcularem els coeficients c_i:

$$\begin{cases} 5^2 = 25 = 12 = 12^{c_0} & \Rightarrow \quad c_0 = 1, \\ (5 \cdot 12^{-1})^2 = (5 \cdot 12^{-1})^2 = (5 \cdot 12)^2 = (8)^2 = 12 = 12^{c_1} & \Rightarrow \quad c_1 = 1. \end{cases}$$

Per tant, $\ell = 1 + 1 \cdot 2 = 3 = \log_8(5)$.

4.1. Algorisme de Silver-Pohlig-Hellman.

Prenem un grup G d'ordre n B-llis. Suposem la factorització de $n = p_1^{\alpha_1} \cdot ... \cdot p_r^{\alpha_r}$ donada. Tot i que el problema de la factorització és intractable, hi ha algorismes eficients per trobar-ne la solució en grups llisos (veure la secció 2.3 de [**3**]).

Per trobar el logaritme discret en G, primer es fan una sèrie de precomputacions que ajuden a reduir el cost de la segona part. Aquesta fase consisteix en generar la taula $\{R_{p_i,j}\}$. Per cada p_i de la factorització, es calculen les p_i-arrels de la unitat $R_{p_i,j} = g^{j \cdot n/p_i}\ \forall\ j \in \{0, 1, ..., p_i - 1\}$. Finalment, s'ordena la taula per a cada p_i de manera que $R_{p_i,j} < R_{p_i,j+1}\ \forall\ j$.

En segon lloc, es procedeix a caluclar el logaritme discret que estem buscant. Es vol trobar k tal que $h = g^k$, on k està determinat mòdul n. Pel teorema xinès del residu podem trobar la solució k resolent el sistema xinès de congruències:

$$\begin{cases} k \equiv x_1 \mod p_1^{\alpha_1}, \\ \quad | \\ k \equiv x_r \mod p_r^{\alpha_r}. \end{cases}$$

Per tant, només hem de trobar k mod $p_i^{\alpha_i}$. Per fer-ho procedim de la següent manera:

Sabem que $k = c_0 + c_1 p + ... + c_{\alpha-1} p^{\alpha-1}$ amb $0 \leqslant c_i < p$. Llavors, per calcular c_i es busca el subíndex j que dóna el match de $h_i^{n/p^{i+1}}$ a la taula $R_{p,j}$.

Per trobar c_0:

$$h^{n/p} = (g^k)^{n/p} = g^{kn/p} = g^{c_0 n/p} = R_{p,c_0}.$$

Per trobar c_1:

$$h_1 = \frac{h}{g^{c_0}} = g^{k-c_0}, \quad \widetilde{k} = k - c_0.$$

$$h_1^{n/p^2} = (g^{\widetilde{k}})^{n/p^2} = g^{\widetilde{k}n/p^2} = g^{c_1 n/p} = R_{p,c_1}.$$

Per trobar c_i:

$$h_i = \frac{h}{g^{c_0+c_1 p+...+c_{i-1}p^{i-1}}} = g^{k-(c_0+c_1 p+...+c_{i-1}p^{i-1})}, \quad \widetilde{k} = k-(c_0+c_1 p+...+c_{i-1}p^{i-1}).$$

$$h_i^{n/p^{i+1}} = (g^{\widetilde{k}})^{n/p^{i+1}} = g^{\widetilde{k}n/p^{i+1}} = g^{c_i n/p} = R_{p,c_i}.$$

Cost de l'algorisme

TEOREMA 2.10. *L'algorisme de Silver Pohlig Hellman té un cost en temps de* $O(B \log B \log n)$

DEMOSTRACIÓ.

Donada la factorització de l'ordre del grup $n = p_1^{\alpha_1} \cdot ... \cdot p_r^{\alpha_r}$, tenim que:

$$P = \#\text{ Primers al factoritzar (comptant repeticions)} = O(\log n).$$

Per trobar el cost de l'algorisme primer mirem la complexitat de fer la precomputació. Per cada p s'han de calcular les arrels de la unitat i guardar-les en $R_{p,j}$ (un total de p arrels amb cost $O(p)$) i després ordenar els elements obtinguts (amb cost $O(p \log p)$ usant un algorisme eficient). Per tant, el cost de fer això és $p + p \log p = O(p \log p)$. Llavors, el cost per a tots els p_i és:

$$\sum_{i=1}^{r} p_i \log p_i \leqslant \sum_{i=1}^{r} B \log B = O(B \log B \log n),$$

ja que $r \leqslant P = O(\log n)$.

El següent pas és calcular els logaritmes per a cadascuna de les potències de primers. Hem de buscar cada c_i en la taula $R_{p,j}$ amb un cost de $\log p_i = O(\log B)$. Com que tenim α_i coeficients per buscar en la taula, el cost total d'aquest pas és:

$$\sum_{i=1}^{r} \alpha_i \log p_i = \log B \sum_{i=1}^{r} \alpha_i = \log B \cdot P = O(\log B \log n).$$

Per tant, com que aplicar el teorema xinès del residu té un cost inferior a la resta de passos podem establir que el cost asimptòtic de l'algorisme és: $\text{cost}(n, B) = O(B \log B \log n)$. □

Observació 2.3. Amb l'algorisme de Silver-Pohlig-Hellman reduïm el cost de calcular el logaritme discret fent servir la factorització de l'ordre del grup G i que el grup és llis. En criptografia es trien grups de manera que no es pugui aplicar aquesta reducció, ja que sinó es facilitaria el càlcul del logaritme discret comprometent la seguretat dels mètodes criptogràfics. Així doncs, es prenen grups on l'ordre és primer o gairebé primer (i.e. té algun factor molt gran comparat amb $|G|$).

Capítol 3
Algorisme de càlcul d'índex

L'algorisme del càlcul d'índex aprofita l'estructura que tenen certs grups cíclics, on és possible trobar un conjunt d'elements generadors que s'anomenen base de factors. Aquest és el cas, per exemple, dels grups multiplicatius de cossos finits. En canvi, aquest mètode no és aplicable per a trobar el logaritme discret sobre corbes el·líptiques, ja que si existeix no se sap trobar una base de factors per a aquest grup.

Per tant, l'objectiu de l'algorisme és calcular el logaritme discret partint d'una base de factors $\mathcal{B}$ de G. Aquesta base de factors és un conjunt d'elements $\{p_1, ..., p_r\} \subset G$ que generen una quantitat elevada d'elements de G i on és fàcil comprovar si un element està generat per $\mathcal{B}$. Direm que g està generat per $\mathcal{B}$, o també que és $\mathcal{B}$-llis, si es poden trobar de forma eficient uns α_i tal que:

$$g = \prod_{p_i \in \mathcal{B}} p_i^{\alpha_i}.$$

Per exemple, veiem com es defineixen les bases de factors pels grups multiplicatius. Per al grup $G = \mathbb{F}_p^*$ la base de factors $\mathcal{B}$ que es fa servir és la següent:

Definició 3.1. *Agafem el conjunt $\mathcal{B}$ de $\mathbb{F}_p^*$ com els primers menors que un fita B ($B \ll p$).*

$$\mathcal{B} = \{p_1, ..., p_r \mid p_i \text{ és primer i } p_i \leqslant B\}.$$

Donat un element $h \in \mathbb{F}_p^*$ per mirar si $h = p_1^{\alpha_1} \cdot ... \cdot p_r^{\alpha_r}$ es mira si és B-llis.

Per un grup $G = \mathbb{F}_q^*$ amb $q = p^t$, que són els polinomis de grau menor que t, la base de factors $\mathcal{B}$ és de la forma següent:

Definició 3.2. *Agafem el conjunt $\mathcal{B}$ de $\mathbb{F}_q^*$ com els polinomis primers de grau menor que una fita B ($B \ll t$).*

$$\mathcal{B} = \{f_1, ..., f_r \mid f_i \text{ és un polinomi irreductible amb } \deg(f_i) \leqslant B\}.$$

En aquests dos casos triant B de forma adequada s'aconsegueix que una gran quantitat d'elements del grup siguin B-llisos. La tria de B repercuteix de forma significativa en la complexitat de l'algorisme.

Veiem l'estructura que segueix, en general, l'algorisme del càlcul d'índex. L'algorisme consta de tres passos. En els dos primers es fan una serie de precomputacions que no s'han de repetir en cas de voler calcular més d'un logaritme. Finalment, en el tercer pas es procedeix a calcular el logaritme discret fent servir la informació obtinguda en els passos 1 i 2.

1. Trobar relacions entre els logaritmes dels elements de $\mathcal{B}$.

   Per fer-ho prenem $u \in \{0, 1, 2, ..., n-1\}$ aleatòriament, calculem g^u i mirem si és B-llis; en aquest cas tenim una relació del tipus:

   $$g^u = p_1^{\alpha_1} \cdot ... \cdot p_r^{\alpha_r}.$$

   Llavors, aplicant logaritmes a ambdós costats de la igualtat obtenim:

   $$u = \log_g(g^u) = \alpha_1 \log_g p_1 + ... + \alpha_r \log_g p_r. \tag{11}$$

   Apliquem aquest mètode fins a trobar r o més relacions d'aquest tipus que siguin linealment independents.

2. Solucionar el sistema lineal generat en el pas 1.

   Tenim un sistema lineal amb r variables, els $\log_g(p_i)$. Per tant, un cop tenim r equacions independents del tipus (11) el sistema es pot solucionar amb tècniques d'àlgebra lineal. D'aquesta manera es troben tots els logaritmes discrets dels elements de $\mathcal{B}$.

3. Trobar el logaritme discret de $h = g^k$.

   Per fer-ho cal una equació que relacioni h amb els elements de $\mathcal{B}$. Es pren $e \in \{0, 1, 2, ..., n-1\}$ aleatori fins a trobar una relació del tipus:

   $$h \cdot g^e = p_1^{e_1} \cdot ... \cdot p_r^{e_r}.$$

   Llavors, el valor del logaritme serà:

   $$k = e_1 \log_g(p_1) + ... + e_r \log_g(p_r) - e.$$

   on tots els logaritmes i coeficients són coneguts.

Veiem, doncs, amb més detall com procedir per realitzar els tres passos exposats prèviament pel cas particular de $G = \mathbb{F}_p^*$, on $|G| = n = p - 1$.

Per aquest grup en particular, farem algunes modificacions en l'algorisme general per tal d'optimitzar-lo. Cal tenir en compte que a l'hora de buscar elements

llisos cal fer-ho de forma aleatòria, ja que no tenim una propietat per crear elements llisos sinó que només tenim mètodes per comprovar si un element ho és o no.

Així doncs, en el primer pas procedirem a prendre u de forma aleatòria i comprovar si g^u genera un element llis amb el Polynomial Sieve method. En cas de tenir un element llis, farem servir l'algorisme de la divisió (Trial Division) per trobar els divisors de g^u. Expliquem, doncs, a continuació aquests dos algorismes.

L'algorisme Trial Division procedeix dividint l'element h pels primers $p_i \in \mathcal{B}$ i ens guardem els elements pels quals la divisió dóna exacta fins que tenim quocient 1. En cas que un dels primers estigui per sobre la fita B l'element no serà llis. El mètode té complexitat $O(\text{card}(\mathcal{B}))$ si els exponents k_i són prou petits i com a molt el cost pot ser $O(\text{card}(\mathcal{B}) + \log_2(h))$, ja que $\log_2(h)$ és el màxim nombre de factors que podem tenir comptant repeticions. (Per més detall veure l'algorisme B.8 de l'apèndix 2).

El mètode Polynomial Sieve (mètode de garbell de polinomis) ens permet determinar de forma conjunta si un nombre gran d'elements són llisos o no. Comprovem amb una sola execució de l'algorisme, si el conjunt d'imatges d'un polinomi $f(x) = a_n x^n + ... + a_1 x + a_0$ amb $x \in [c, d)$ són elements llisos. (Per més detall veure l'algorisme B.9 de l'apèndix 2).

Per aplicar el mètode del garbell de polinomis cal modificar el pas 1 del càlcul d'índex. Això es deu al fet que al buscar $g^u = p_1^{\alpha_1} \cdot ... \cdot p_r^{\alpha_r}$ no tenim un polinomi al que aplicar el garbell. Per aquest motiu és recomanable procedir de la forma següent pel càlcul.

1. Definim $H = \lceil \sqrt{p} \rceil$ i $c_1,\ c_2$ petits i no negatius. Llavors, sabem que:

$$H^2 = (\lceil \sqrt{p} \rceil)^2 \leqslant (\sqrt{p} + 1)^2 = p + \underbrace{2\sqrt{p} + 1}_{J} = p + J, \tag{12}$$

   on $J = O(\sqrt{p})$.

2. Calculem el producte $(H + c_1)(H + c_2) = H^2 + (c_1 + c_2)H + c_1 \cdot c_2 = J + (c_1 + c_2)H + c_1 \cdot c_2 \pmod{p} = O(\sqrt{p})$.

3. Per c_1 fixat usem el Polynomial Sieve per a c_2 lliure. Tenim un polinomi de grau 1 en c_2:

$$f(c_2) = (H + c_1) \cdot c_2 + (J + c_1 H).$$

4. Pels elements que superen el test tindrem que:

$$(H + c_1)(H + c_2) = p_1^{e_1} \cdot ... \cdot p_r^{e_r}.$$

Prenent logaritmes obtenim:

$$\log_g(H + c_1) + \log_g(H + c_2) = e_1 \log_g(p_1) + ... + e_r \log_g(p_r).$$

Cal observar que les variables de sistema que hem de resoldre són $\log_g(p_1), ...,$ $\log_g(p_r)$ i també $\log_g(H + c_1),\ \log_g(H + c_2)$. Així doncs, al afegir una nova equació al problema també hi afegim noves variables. Les variables que s'afegeixen augmenten més lentament que les equacions trobades, ja que amb una combinació nova de variables c_1, c_2 ja usades obtenim una nova equació. Per tant, acabem trobant més equacions que variables tenim.

El pas següent, consisteix en solucionar el sistema lineal obtingut amb $\log_g(p_i)$ i $\log_g(H + c_i)$ com a variables. D'aquesta forma passem a conèixer el valor del logaritme dels factors de $\mathcal{B}$.

Per últim necessitem obtenir una relació entre l'element h del que busquem el logaritme i els elements de $\mathcal{B}$. Per fer-ho triem $U \geqslant B$ de forma adequada (veure l'anàlisi del cost de l'algorisme). Llavors, elegim de forma aleatòria exponents w fins a aconseguir que $r = h \cdot g^w$ sigui U-llis. Cal fer notar que per comprovar si r és U-llis no es pot fer servir l'algorisme del Polinomial Sieve, ja que no podem representar r com a imatge d'un polinomi. Es poden fer servir mètodes com mètode de factorització de la ρ o el mètode de factorització de corbes el·lìptiques. En aquest treball ens centrem en l'estudi del logaritme discret i, per tant, no s'explicaran aquests dos mètodes de factorització d'enters.

Un cop trobat w tenim la relació següent:

$$h \cdot g^w = p_1^{k_1} \cdot ... \cdot p_s^{k_s}.$$

On tenim que $p_i \leqslant U$. Llavors, per als primer menors que B ja sabem el valor del seu logaritme gràcies a la precomputació i pels primers $B < p_i \leqslant U$ necessitem fer alguns càlculs per trobar-ne el logaritme.

Si tenim p_i U-llis, però no B-llis prenem $u = \left\lceil \frac{\sqrt{p}}{p_i} \right\rceil$ i l'augmentem variant c en $\widetilde{u} = u + c$ fins a obtenir $\widetilde{u}$ B-llis. Llavors, $\log_g(\widetilde{u})$ és conegut a partir de la factorització. Posteriorment, busquem una relació del tipus $z = \widetilde{u} \cdot v \cdot p_i$ amb z B-llis, obtinguda variant v. Comencem prenent $v = \lceil \sqrt{p} \rceil$ i l'augmentem fins a trobar z B-llis. Fem notar que $\log_g(z)$ és conegut per ser llis i $\log_g(v) = \log_g(H + c_i)$ conegut de la precomputació. Per tant,

$$\log_g(p_i) = \log_g(z) - \log_g(v) - \log_g(\widetilde{u}).$$

Notem que tant $\widetilde{u}$ com v es poden trobar fent servir l'algorisme de Polynomial Sieve. Per usar-lo només cal definir els polinomis en c següents:

$$p(c) = \widetilde{u} = c + \left\lceil \frac{\sqrt{p}}{p_i} \right\rceil \quad \text{i} \quad q(c) = z = \widetilde{u} \cdot p_i \cdot c + \widetilde{u} \cdot p_i \cdot \lceil \sqrt{p} \rceil.$$

Cost de l'algorisme

El primer que cal tenir en compte és que el cost de l'algorisme del càlcul d'índex depen significativament de la tria de la fita B. Si B és molt petit el cost de comprovar si un element és llis i el cost de la resolució del sistema lineal serà molt baix. Tot i així, trobar elements llisos es complica enormement, ja que hi haurà molts pocs nombres a l'interval $[1, p-1]$ que siguin producte de primers menors que B. Recíprocament, si B és molt gran hi haurà molts elements llisos a l'interval, però comprovar si un element és llis i resoldre el sistema serà molt complicat. Per tant, és convenient triar una fita B de tamany mitjà.

Per trobar B adequat cal estudiar la probabilitat que un element $r \in [1, p-1]$ sigui B-llis.

DEFINICIÓ 3.3. *Siguin $B, x \in [1, p-1]$. Definim el nombre d'elements B-llisos menors que x com:*

$$\Psi(x, B) = \#\{r \mid 1 \leqslant r \leqslant x \text{ amb } r \text{ } B\text{-llis}\}.$$

Es coneixen diferents estimadors de $\Psi(x, B)$. En particular, aquí farem servir una fita introduïda per Dickman (1930):

$$\Psi(x, x^{1/u}) \sim x\rho(u). \tag{13}$$

Definida per $u > 0$ fixada i amb $\rho(u)$ la funció de Dickman solució de l'equació diferencial:

$$\begin{cases} u\rho'(u) + \rho(u-1) = 0 & (u > 1), \\ \rho(u) = 1 & (0 \leqslant u \leqslant 1). \end{cases} \quad \Rightarrow \quad \text{Solució: } \rho(u) \sim u^{-u+o(u)}.$$

En concret, pel cas en que ens trobem $\Psi(x, B) = \Psi(x, x^{1/u})$ si prenem $u = \frac{\log x}{\log B}$.

Llavors, la probabilitat que un element de $\mathbb{F}_p^*$ sigui B-llis és:

$$\Pr(r \text{ aleatòri sigui } B\text{-llis}) = \frac{\Psi(x, B)}{x} \sim u^{-u+o(u)},$$

per tant, el nombre esperat de tries d'enters per trobar-ne un de B-llis és:

$$\frac{1}{\Pr(r \text{ } B\text{-llis})} = \frac{1}{u^{-u}} = u^u. \tag{14}$$

Per veure quin és el valor òptim per a B definim la següent funció:

$$L_p[s; c] = e^{c(\log p)^s (\log\log p)^{1-s}}. \tag{15}$$

Aquesta funció és útil per estudiar complexitats que estan entre un polinomi i una exponencial. Per $s = 0$ és un polinomi en els bits de p ($L_p[0; c] = (\log p)^c$) i per $s = 1$ és una exponencial ($L_p[1; c] = e^{c \log p}$).

Pel teorema dels nombres primers sabem que $|\mathcal{B}| \sim \frac{B}{\log B}$. Llavors si busquem un

nombre de relacions amb elements B-llisos (del pas 1) múltiple de la mida de la base de factors, necessitem $\frac{D \cdot B}{\log B}$ elements B-llisos amb D una constant. Usant (14) veiem que el nombre total d'elements pels quals hem de comprovar si són llisos és:

$$\frac{DB}{\log B} \cdot u^u.$$

Per dur a terme aquestes comprovacions es fa servir el mètode del Polynomial Sieve. Es pot veure en [**10**] que el cost d'aplicar el garbell a un interval de longitud L és $O(|\mathcal{B}|(1+\log B)^{o(1)} + L \log\log B)$. Per tant, el temps per a cada enter de l'interval és $\log\log B$. Així doncs, el temps necessari per crear el sistema lineal és:

$$T(B) = \frac{DBu^u}{\log B} \log\log B.$$

Prenem logaritmes en l'expresió anterior:

$$t(B) = \log T(B) = \log D + \log B + u \log u - \log\log B + \log\log\log B.$$

Precindim dels termes $\log\log B,\ \log\log\log B$ que estan dominats per $\log B$. Llavors, derivant la expresió i usant $u = \frac{\log x}{\log B}$ arribem a:

$$\frac{dt}{dB} = \frac{1}{B} + \frac{du}{dB}(\log u + 1) = \frac{1}{B} - \frac{\log x}{B(\log B)^2}(1 + \log\log x - \log\log B).$$

Prenem aquesta derivada igual a zero per trobar el valor òptim:

$$(\log B)^2 = \log x(1 + \log\log x - \log\log B). \tag{16}$$

A més, sabem que $1 < \log\log B < \log\log x$ ja que B ha de ser prou gran per trobar elements llisos amb facilitat (d'on es dedueix la primera desigualtat) i $B < x$ perquè sinó tot element és llis. Per tant, usant aquestes desigualtats és clar que:

$$\log x < \log x(1 + \log\log x - \log\log B) < \log x \log\log x. \tag{17}$$

Combinant les expresions (16) i (17) obtenim que el valor òptim de B satisfà:

$$e^{\sqrt{\log x}} < B < e^{\sqrt{\log x \log\log x}}.$$

Prenent $x = p$ la tria òptima de B és de la forma $B = L_p[\frac{1}{2}; c_B + o(1)]$ per una certa constant c_B.

THEOREMA 3.1. *Donat $G = \mathbb{F}_p^*$ i agafant B de la forma descrita, el cost de l'algorisme del càlcul d'índex és:*

$$\text{cost}(p) = O(e^{(1+o(1))(\sqrt{\log p \log\log p})}).$$

DEMOSTRACIÓ.

Per obtenir el cost de l'algorisme comencem mirant el cost de la precomputació, que és el pas que més temps necessita. Un cop fet això només cal veure que el cost de calcular un logaritme (pas 3) està dominat pel cost de la precomputació.

Definim $B = L_p[s_B; c_B + o(1)]$ per unes certes constants s_B, c_B. Hem vist que el B òptim és d'aquesta forma i, encara més, que s_B és $1/2$ com tornarem a veure.

Prenem $0 \leqslant c_1 < c_2 \leqslant C$ amb $C = L_p[s_C; c_C + o(1)]$ per unes constants s_C, c_C adequades. Per aquest $c_1,\ c_2$ mirem si $(H+c_1)(H+c_2)$ és llis. El total de parells (c_1, c_2) pels que comprovem si l'element és llis són:

$$\sum_{c_2=1}^{C} c_2 = \frac{C \cdot (C+1)}{2} \sim \frac{C^2}{2} = O(L_p[s_C; 2c_C + o(1)]). \tag{18}$$

Cal veure quines condicions han de complir B, C per tal de trobar el suficient nombre d'elements llisos i poder resoldre el sistema lineal del pas 2.

El nombre de relacions amb elements llisos que cal trobar és $B+C$, ja que tenim $C+1$ variables del tipus $(H + c_i)$ i $|\mathcal{B}| \leqslant B - 1$ variables en $\mathcal{B}$. Així doncs, obtenim:

$$B+C = L_p[s_B; c_B+o(1)]+L_p[s_C; c_C+o(1)] = L_p[\max\{s_B, s_C\}; \max\{c_B, c_C\}+o(1)].$$

On la igualtat només es dóna si $s_B = s_C$, que veurem que és el nostre cas. Sinó la segona igualtat de l'equació anterior seria una fita superior.

Prenem P_q la probabilitat que un element aleatori sigui llis. Anem a veure que $P_q = L_p[s_q; c_q + o(1)]$ per unes constants adequades.

Prenem $x = (H + c_1)(H + c_2) \pmod p$, on hem definit $H = \left\lceil L_p[1; \frac{1}{2}] \right\rceil = \left\lceil p^{1/2} \right\rceil$ i $J = H^2 - p \leqslant 2H$. Això ens permet fitar x:

$$x = J + (c_1 + c_2)H + c_1c_2 \leqslant (2 + c_1 + c_2)H + c_1c_2 \leqslant (2 + 2C)H + C^2 =$$

$$= O(L_p[s_C; c_C + o(1)] \cdot L_p[1; \frac{1}{2}] + L_p[s_C; 2c_C + o(1)]) = O(L_p[1; \frac{1}{2} + o(1)]).$$

On la última igualtat es dedueix del fet que $L_p[1; \cdot]$ és exponencial i $L_p[s_C; \cdot]$ és subexponencial. Per tant, tots els elements que agafem a l'algorisme estan per sota de $\overline{x} = L_p[1; \frac{1}{2} + o(1)]$.

Sabent tot això, veiem quina és la probabilitat que un element sigui llis. On tenim que:

$$u = \frac{\log \overline{x}}{\log B},\ B = L_p[s_B; c_B + o(1)],\ \overline{x} = L_p[1; \frac{1}{2} + o(1)].$$

$$\begin{aligned} P_q \sim u^{-u+o(u)} &= \left(\frac{(\frac{1}{2}+o(1))\log p}{(c_B+o(1))(\log p)^{s_B}(\log\log p)^{1-s_B}}\right)^{-\frac{(\frac{1}{2}+o(1))\log p}{(c_B+o(1))(\log p)^{s_B}(\log\log p)^{1-s_B}}+o(u)} \\ &= e^{-(1-s_B)(\frac{1/2+o(1)}{c_B+o(1)})(\log p)^{1-s_B}(\log\log p)^{-1+s_B}(\log\log p+O(\log\log\log p))} \\ &= e^{-(1-s_B)(\frac{1/2+o(1)}{c_B+o(1)}+o(1))(\log p)^{1-s_B}(\log\log p)^{s_B}} \\ &= L_p[1-s_B; -\tfrac{1-s_B}{2c_B}+o(1)]. \end{aligned}$$

Un cop trobada la probabilitat P_q podem donar una condició per tal que el sistema lineal obtingut al pas 1 tingui solució. On cal que l'esperança del nombre d'elements llisos trobats, calculada usant (18), sigui major dels que necessitem:

$$\frac{1}{2}C^2 P_q \geqslant B + C. \tag{19}$$

Llavors, la condició (19) de tenir suficients elements llisos per construir el sistema lineal es pot expressar de la següent forma:

(20)

$$L_p[s_C; 2c_C+o(1)] \geqslant L_p[\max\{s_B, s_C\}; \max\{c_B, c_C\}+o(1)]\cdot L_p[1-s_B; \frac{1-s_B}{2c_B}+o(1)].$$

On hem canviat P_q de costat, canviant el signe al segon terme de L_p. Com que els exponents dominen la desigualtat, això implica que:

$$s_C \geqslant \max\{s_B, s_C, 1-s_B\}. \tag{21}$$

Aquesta condició ens garanteix poder resoldre el sistema lineal amb les constants C i B triades (i.e. ens diu que tindrem els suficients elements llisos). Ara, anem a mirar quin és el cost de tots els passos de l'algorisme i fixarem les constants B, C que el minimitzen, complint sempre la inequació (21).

El mètode del garbell de polinomis es fa prenent c_1 fixat i agafant un polinomi de grau 1 en c_2. Per tant, hem d'aplicar l'algorisme C cops, un per cada c_1, amb un cost total de:

$$\begin{aligned} C\cdot \text{cost}_{\text{Polynomial Sieve}} &= C\cdot O(|\mathcal{B}|(1+\log B)^{o(1)} + C\log\log B) \\ &= L_p[s_C; c_C]\cdot(L_p[s_B; c_B] + L_p[s_C; c_C]) \\ &= L_p[\max\{s_B, s_C\}; c_C + \max\{c_B, c_C\} + o(1)). \end{aligned}$$

La resolució del sistema lineal té cost cúbic en la quantitat de variables:

$$(B+C)^3 = L_p[\max\{s_B, s_C\}; \max\{3c_B, 3c_C\} + o(1)].$$

Com el cost del garbell és menor que el del sistema lineal, el total de temps necessari per fer la precomputació és:

$$L_p[\max\{s_B, s_C\}; \max\{3c_B, 3c_C\} + o(1)]. \tag{22}$$

Per minimitzar el temps hem de trobar s_B i s_C que minimitzin el valor de $\max\{s_B, s_C\}$ sense trencar la condició de compatibilitat (21). D'on es dedueix que $s_B = s_C = \frac{1}{2}$ ja que si $s_B < 1/2 \Rightarrow \max\{s_B, s_C\} \geqslant s_C \geqslant 1 - s_B > 1/2$ i igualment si $s_B > 1/2$.

Llavors, podem reescriure la equació (20) amb la nova informació obtinguda:

$$L_p[\frac{1}{2}; 2c_C + o(1)] \geqslant L_p[\frac{1}{2}; \max\{c_B, c_C\} + o(1)] \cdot L_p[\frac{1}{2}; \frac{1}{4c_B} + o(1)]. \tag{23}$$

Com que a la desigualtat tots els termes tenen el mateix exponent, deduïm una altra condició per tenir suficients equacions en el sistema lineal:

$$2c_C \geqslant \max\{c_B, c_C\} + \frac{1}{4c_B}. \tag{24}$$

Llavors, volem minimitzar $\max\{3c_B, 3c_C\}$ per reduir el cost (22) de la precomputació. Tenint en compte la desigualtat (24) deduim que $c_B = c_C = \frac{1}{2}$. S'obté del fet que si reescrivim la desigualtat com $2c_C - \frac{1}{4c_B} \geqslant \max\{c_B, c_C\}$ veiem que el primer terme de l'esquerra augmenta amb c_C i el segon augmenta amb c_B més ràpidament del que ho fan a la dreta. Per tant, si reduïm un dels valors l'altre ha d'augmentar i empitjora el resultat que volem minimitzar. Per tant, cal que $c_B = c_C$ i el valor mínim que compleix (24) és $\frac{1}{2}$.

Així doncs, el temps total necessari per fer la precomputació és:

$$O(L_p[\frac{1}{2}; \frac{3}{2} + o(1)]).$$

Ens queda per veure quin és el cost del pas 3. El primer que cal fer és definir $U \geqslant B$ i trobar w tal que $h \cdot g^w \pmod p$ sigui U-llis. Prenem $U = L_p[s_U; c_U + o(1)]$ i la probabilitat de trobar w adecuat $P_w = L_p[s_w; c_w + o(1)]$. Llavors, com tenim $p = L_p[1; 1]$ es pot procedir de la mateixa forma que amb P_q per veure que:

$$P_w = \frac{\Psi(p, U)}{p} = L_p[1 - s_U; -\frac{1 - s_U}{c_U} + o(1)]$$

Per tal que aquest pas tingui complexitat per sota de la precomputació es fa servir el mètode de factorització amb corbes el·líptiques. La complexitat d'aquest mètode per decidir si un element és llis és $O(e^{\sqrt{(2+o(1))\log U \log\log U}}(\log p)^2)$ que amb notació L_p s'escriu com:

$$L_p[\frac{s_U}{2}; \sqrt{2 s_U c_U} + o(1)] \cdot L_p[0; 2] = L_p[\frac{s_U}{2}; \sqrt{2 s_U c_U} + o(1)].$$

Com que el nombre d'elements que cal mirar per trobar w adecuat és $1/P_w$, el temps total per trobar w és:

$$L_p[1 - s_U; \frac{1 - s_U}{c_U} + o(1)] \cdot L_p[\frac{s_U}{2}; \sqrt{2 s_U c_U} + o(1)] = L_p[\max\{1 - s_U, \frac{s_U}{2}\}; K]. \tag{25}$$

En aquest cas, volem minimitzar $\max\{1-s_U, \frac{s_U}{2}\}$ d'on s'obté $s_U = \frac{2}{3}$. Es dedueix del fet que un terme augmenta i l'altre decreix en s_U, per tant, el resultat més petit es dóna quan hi ha igualtat entre els dos factors (sinó un dels dos es fa més gran).

Substituint s_U en (25) obtenim una expressió per K:

$$L_p[\frac{1}{3}; \frac{1}{3c_U} + o(1)] \cdot L_p[\frac{1}{3}; 2\sqrt{\frac{c_U}{3}} + o(1)] = L_p[\frac{1}{3}; \frac{1}{3c_U} + 2\sqrt{\frac{c_U}{3}}].$$

Per trobar U òptim cal minimitzar $\frac{1}{3c_U} + 2\sqrt{\frac{c_U}{3}}$ d'on s'obté $c_U = \left(\frac{1}{3}\right)^{1/3}$ derivant l'expressió. Així doncs, cal triar U de la forma següent:

$$U = L_p[\frac{2}{3}; \left(\frac{1}{3}\right)^{1/3} + o(1)].$$

Finalment, el temps total d'aquest pas és:

$$L_p[\frac{1}{3}; 3^{1/3} + o(1)].$$

Per últim, cal passar d'un element U-llis a un de B-llis. Pels primers de mida inferior a B ja hem acabat. Llavors, pel primers de mida mitjana hem de fer algunes computacions extres. Sigui m un primer U-llis però no B-llis. Busquem $u > \sqrt{p}/m$ tal que és B-llis. La probabilitat que en una tria aleatòria de u sigui B-llis és $P_q = L_p[\frac{1}{2}; -\frac{1}{2}]$. Per trobar-ne un usem el Polynomial Sieve que tarda un temps de $L_p[\frac{1}{2}; 1 + o(1)]$.

Per últim cal trobar $v > \sqrt{p}$ tal que $u \cdot v \cdot m \pmod{p}$ sigui B-llis. Tenim que v és de l'ordre de $L_p[1; \frac{1}{2}]$. Que ens porta al mateix cost asimptòtic que la precomputació:

$$L_p[\frac{1}{2}; \frac{1}{2} + o(1)].$$

Aquests dos últims passos s'han de realitzar per cada factor que no sigui B-llis. Per tant, hi haurà un màxim de $\log_B U = \frac{\log U}{\log B}$ d'aquest tipus. Pel cas de $s_U = 2/3$ tenim que n'hi ha com a molt $O((\log p)^{1/6})$ i un factor polinòmic no afecta al cost comparativament amb els altres passos.

Per tant, el cost de l'algorisme és el cost de la precomputació, que és el cost de l'enunciat del teorema. □

Capítol 4
Logaritme discret i factorització

Els protocols criptogràfics de clau pública fan servir una funció unidireccional (veure la secció 3.1) per tal de crear el parell (k_s, k_p). D'aquesta manera obtenir k_p a partir de k_s té cost polinòmic i, en canvi, obtenir k_s a partir de k_p implica solucionar un problema que es considera intractable. En els mètodes actuals, el problema que s'ha de solucionar per obtenir k_s és la factorització d'enters o el logaritme discret, depenent de la funció unidireccional usada.

Com que la component pública k_p de la clau és coneguda per tothom, és de vital importancia que sigui intractable recuperar k_s a partir d'ella. Així doncs, és interessant veure quin dels dos mètodes, si la factorització d'enters o el logaritme discret, és millor per donar seguretat al sistema criptogràfic.

Si apliquem el logaritme discret en grups multiplicatius podríem pensar que els dos problemes tenen la mateixa complexitat. Això es deu al fet que el cost dels algorismes més eficients per factoritzar i per resoldre el logaritme discret (sobre el grup $\mathbb{F}_q^*$) és el mateix. El cost d'aquests algorismes és subexponencial, el logaritme discret en $\mathbb{F}_q^*$ es resol amb l'algorisme del càlcul d'índex que és l'anàleg de l'algorisme de bases de factors per a la factorització d'enters.

A més, hi ha algorismes que van ser dissenyats per solucionar un dels dos problemes, però es poden adaptar per solucionar l'altre problema. Per exemple, l'algorisme ρ de Pollard es fa servir per solucionar tant un problema com l'altre.

A pesar d'aquestes similituds, quan prenem el logaritme discret sobre corbes el·líptiques veiem les diferències entre els dos problemes. El cost de trobar el logaritme discret en aquests grups és exponencial a diferència de la factorització i el logaritme a $\mathbb{F}_p^*$. Per tant, en criptografia és millor usar funcions unidireccionals on obtenir k_s passi per resoldre un logaritme discret sobre corbes el·líptiques.

Hem vist que si l'ordre n del grup G és llis, podem reduir el cost de calcular el logaritme discret. Per fer-ho s'aplica l'algorisme de Pohlig-Hellman. Per tant,

factoritzar n ens ajuda a calcular el logaritme discret reduint-nos a calcular logaritmes sobre els seus factors primers. Tot i així, si alguns factors de n són prou grans això és de poca utilitat.

El teorema 4.1 afirma que si sabem solucionar el problema del logaritme discret de forma eficient, també podrem resoldre el problema de la factorització d'enters.

DEFINICIÓ 4.1. *Es defineix la funció d'Euler com*

$$\varphi(n) = \#(\mathbb{Z}/n\mathbb{Z})^*.$$

TEOREMA 4.1. *Sigui G el grup $(\mathbb{Z}/N\mathbb{Z})^*$. Suposem que existeix un algorisme A que resol el logaritme discret en temps polinòmic. Llavors, existeix un algorisme probabilístic que resol la factorització d'enters en temps polinòmic.*

DEMOSTRACIÓ.

Si N és parell té el factor $p = 2$. Per tant, es pot suposar N senar.

Suposem també que N no és potència d'un primer, ja que sinó $N = p^t$ es pot factoritzar de forma eficient fent:

1. Calcular una aproximació de les $\log_2 N$ primeres arrels reals de N (ja que $t \leqslant \log_2 N$);
2. Per a cadascuna de les arrels comprovar si el resultat obtingut és enter; si ho és, ja hem trobat un factor de N.

Si N no està en cap dels casos previs, tenim que $N = p_1^{\alpha_1} \cdot ... \cdot p_r^{\alpha_r}$ amb $r \geqslant 2$ i per tant:

$$(\mathbb{Z}/N\mathbb{Z})^* \cong (\mathbb{Z}/p_1^{\alpha_1}\mathbb{Z})^* \times ... \times (\mathbb{Z}/p_r^{\alpha_r}\mathbb{Z})^*.$$

Així doncs, $(\mathbb{Z}/N\mathbb{Z})^*$ no és cíclic. Per tant, podem trobar $1 < e < \varphi(N)$ tal que $a^e \equiv 1 \pmod N$. On e és l'ordre de a, complint que $e \mid L = \text{mcm}(\varphi_1, ..., \varphi_r)$, on $\varphi_i = \varphi(p_i^{\alpha_i}) = p_i^{\alpha_i - 1}(p_i - 1)$.

Per passar del logaritme discret a la factorització, ho fem amb dos passos:

Pas 1: Reduir la factorització al càlcul d'ordres. Si sabem trobar l'ordre dels elements de $(\mathbb{Z}/N\mathbb{Z})^*$, llavors sabem trobar la factorització de N.

Pas 2: Reduir el càlcul d'ordres al logaritme discret en $(\mathbb{Z}/N\mathbb{Z})^*$. Si sabem solucionar el logaritme discret mòdul N, llavors sabem calcular l'ordre dels elements del grup.

Pas 1:

Definim el conjunt següent:

$$K = \{x \in (\mathbb{Z}/N\mathbb{Z})^* \mid x^{L/2} \equiv \pm 1 \bmod N\}.$$

Veiem que $K \neq (\mathbb{Z}/N\mathbb{Z})^*$:

Definim $e_2(k)$ per $k \in \mathbb{Z}$ com l'exponent de 2 en la factorització de k. Ordenem els p_i de manera que $e_2(\varphi_1) \geqslant e_2(\varphi_i)$. D'on s'obté que $e_2(\varphi_1) = e_2(L)$. Com que cada $(\mathbb{Z}/p_i^{\alpha_i}\mathbb{Z})^*$ és cíclic en triem generadors $\{g_1, ..., g_r\}$. Es tria $a \in (\mathbb{Z}/N\mathbb{Z})^*$ expressat en coordenades com $a = (g_1, g_2^2, ..., g_r^2) \Rightarrow a^{L/2} = (g_1^{L/2}, g_2^L, ..., g_r^L) = (g_1^{L/2}, 1, ..., 1)$. Com que g_1 té ordre φ_1 llavors $L/2$ no pot ser múltiple de φ_1 ja que $e_2(L/2) = e_2(\varphi_1) - 1$ (es redueix l'exponent del 2). Per tant, $a^{L/2} \not\equiv \pm 1 \Rightarrow a \notin K$.

Com que K és un subgrup tenim que com a mínim la meitat dels elements de $(\mathbb{Z}/N\mathbb{Z})^*$ no estan en K.

Creem la seqüència que farem servir per factoritzar N:

Prenem $x \in (\mathbb{Z}/N\mathbb{Z})^*$ tal que $x \notin K$. Sabem que $x^m \equiv 1 \pmod N$, per un cert m. Creem la seqüència $x^m, x^{m/2}, x^{m/4}, ..., x^{m/2^{e_2(m)}}$.

Veiem que hi ha un k tal que $x^{m/2^k} \equiv 1$ i $x^{m/2^{k+1}} \not\equiv 1$:

Suposem que l'ordre de x és s. Llavors, $L = c_0 \cdot s$ (ja que $x^L \equiv 1$). A més, $x^{L/2} \not\equiv 1$ per tant c_0 és senar. També, $m = c \cdot s$ d'on podem treure tots els 2 de c obtenint $m = 2^{e_2(c)} \cdot c_1 \cdot s$ amb c_1 senar. Per tant, prenent $k = e_2(c)$ tenim que $x^{m/2^k} \equiv x^{c_1 \cdot s} \equiv 1$. Considerem $x^{m/2^{k+1}} \equiv x^{c_1 \cdot s/2} \equiv x^{s/2}$ (ja que si $z^2 \equiv 1$ llavors per d senar tenim que $z^d \equiv z$) i igualment $x^{L/2} \equiv x^{c_0 \cdot s/2} \equiv x^{s/2}$. Podem concloure que:

$$x^{m/2^{k+1}} \equiv x^{L/2} \not\equiv \pm 1.$$

D'aquí es dedueix que $x^{m/2^{k+1}}$ és una arrel de 1. Veiem que podem trobar factors no trivials de N amb $\mathrm{mcd}(x^{m/2^{k+1}} \pm 1, N) \neq 1$:

Definim $z = x^{m/2^{k+1}} \Rightarrow z \not\equiv \pm 1$ i $z^2 \equiv 1$. Per tant, $z^2 - 1 \equiv 0 \Rightarrow N$ divideix $(z-1)(z+1)$. Però no divideix cap dels factors, ja que $z \not\equiv \pm 1$. Així doncs, $\mathrm{mcd}(z \pm 1, N)$ ens dóna els factors de N.

Repetint el procés per cadascun dels factors trobats podem arribar a la factorització de N.

Pas 2:

Per buscar el valor de m es procedeix de la forma següent:

Prenem p primer tal que $\mathrm{mcd}(p, \varphi(N)) = 1$. Llavors, $\exists\ q$ tal que $p \cdot q \equiv 1 \pmod{\varphi(N)} \Rightarrow$ (Fermat) $x^{pq} \equiv x \pmod N \Rightarrow$ Calculem $\log_{x^p} x = q$ i prenem $m = pq - 1$ que compleix $x^m \equiv 1 (\mathrm{mod}\ N)$.

Algorísmicament, al no conèixer $\varphi(N)$ no podem comprovar si p és coprimer amb $\varphi(N)$. Procedim suposant que p compleix la condició i calculem $\log_{x^p} x = q$

llavors comprovem si $x^{pq} \equiv x \pmod{N}$. Com que $\varphi(N) < N$ hi haurà un primer p coprimer amb $\varphi(N)$ entre els $\log N + 1$ primers.

Com que calcular el logaritme hem suposat que té cost polinòmic i ho fem un màxim de $\log N + 1$ cops, el cost total del pas 2 és polinòmic.

El cost del pas 1 depèn de la tria de l'element x tal que ha de complir $x \notin K$ (com a mínim la meitat dels elements). Triant x de forma aleatòria la probabilitat de tenir un x vàlid és $\geqslant 1/2$. Repetint el procés ℓ cops la probabilitat de fallar és $\leqslant (1/2)^\ell$. Un cop trobada una x vàlida la resta del procés té cost polinòmic i, per tant, el cost és polinòmic amb probabilitat tan alta com vulguem (depenent de ℓ). □

Capítol 5
Aplicacions del logaritme discret a la criptografia

Els sistemes criptogràfics de clau pública que s'explicaran en aquest apartat basen la seva seguretat en la dificultat de solucionar el logaritme discret, ja que aquest és un problema que se suposa intractable. Per trobar el logaritme discret sobre grups generals només es coneixen algorismes de l'ordre de l'arrel quadrada (exponencials),tot i que, per a grups multiplicatius de cossos finits es coneixen algorismes subexponencials (algorisme del càlcul d'índex).

En aquesta secció veurem com aplicar el logaritme discret a la criptografia. Quins mètodes hi ha per crear protocols criptogràfics segurs i com s'apliquen. Per fer això cal distingir entre les dues aplicacions bàsiques de la criptografia de clau pública: encriptar i desencriptar per proporcionar confidencialitat i firmar i verificar missatges per proporcionar autenticitat.

1. Algorismes de xifrat per a la transmissió de la clau

Si dues persones es volen comunicar usant criptografia simètrica, les dues han de tenir la mateixa clau k. Per tant, el primer que han de fer és passar-se aquesta clau de forma segura. La criptografia de clau pública va aportar una gran millora per tal de transmetre de la clau k usada en criptografia simètrica. Abans de fer servir criptografia asimètrica es feien servir mètodes molt menys segurs que els actuals.

1.1. Mètodes de criptografia simètrica (Merkle Puzzles).

El sistema de Merkle puzzles fa servir criptografia simètrica per comunicar la clau secreta. Per fer-ho primer es defineixen $P_i \in \{0,1\}^\ell$, $x_i, k_i \in \{0,1\}^m$ amb $\ell \ll m$ i per $i = 1, ..., 2^\ell = n$. Llavors, el mètode per transmetre la clau secreta entre A i B és el següent:

1. A genera $\text{puzzle}_i = E(P_i, \text{“Puzzle \#}x_i \text{ amb clau } k_i\text{”})$, que és el xifrat del text usant la clau P_i, per $i = 1, ..., n$ i els hi envia tots a B.
2. B tria puzzle_j aleatòriament i el soluciona obtenint (x_j, k_j).
3. B envia x_j a A.
4. A busca x_j i es guarda k_j que usaran com a clau secreta.

Observem que els cost d'aplicar aquest protocol és bàsicament el cost de solucionar el puzzle_j ja que la resta de passos tenen cost més baix. La manera de trobar el missatge és provar els P_i fins a trobar l'adequat amb un cost de $O(n)$. En canvi, un adversari que intenti trobar la clau haurà de solucionar tots els puzzles fins a trobar x_j amb un cost de $O(n^2)$.

Es pot demostrar que el millor que es pot aconseguir fent servir criptografia simètrica és tenir un guany quadràtic (i.e. intercanviar la clau té cost $O(n)$ i trencar el sistema té cost $O(n^2)$). Per tant, el Merkle Puzzles és òptim.

1.2. Mètodes de criptografia asimètrica.

A la introducció s'explica el funcionament general d'un mètode criptogràfic de clau pública. En aquest apartat expliquem amb detall dos protocols per intercanviar la clau secreta mitjançant criptografia asimètrica.

Protocol de Diffie-Hellman

Prenem un grup cíclic G i un generador g. Sigui $|G| = n$. Llavors, el protocol de Diffie-Hellman consta dels passos següents:

1. A i B trien aleatòriament $a, b \in \{1, 2, ..., n-1\}$.
2. A li envia g^a a B i B li envia g^b a A.
3. Ambdós computen $g^{ab} = k_{AB}$ (A la calcula fent $(g^b)^a$ i B fent $(g^a)^b$).

Llavors, tant A com B obtenen la clau secreta $k_{AB} = g^{ab}$ que faran servir per comunicar-se via criptografia simètrica.

Observem que usant aquest protocol criptogràfic podem emmagatzemar g^a i g^b a un servidor web i, llavors, no fa falta cap tipus de comunicació entre A i B per obtenir la clau secreta k_{AB}. Sinó que A i B poden anar directament a buscar la g^a i g^b al servidor i computar k_{AB}. Això resulta de gran utilitat si A penja un arxiu encriptat amb $k_{AB} = g^{ab}$ que vol compartir amb B. Llavors, encara que A no estigui disponible per comunicar-se, B pot agafar g^a del servidor web i crear la clau secreta k_{AB} per desencriptar l'arxiu.

Observem també que aquest mètode es pot interpretar en termes de clau pública, format per dos parells de claus. On la clau de A és $(k_p = g^a, k_s = a)$ i la clau de B és $(k_p = g^b, k_s = b)$. Per tant, aquest protocol és una variant dels protocols de clau pública que es defineixen en general.

La seguretat del sistema d'intercanvi de clau es basa en la hipòtesi de Diffie-Hellman.

DEFINICIÓ 5.1 (Hipòtesi de Diffie-Hellman). *És intractable computacionalment obtenir g^{ab} a partir únicament de g^a i g^b.*

OBSERVACIÓ 5.1. Si el problema del logaritme discret fos tractable podríem obtenir la clau del protocol de Diffie-Hellman en temps polinòmic. Per obtenir la clau només necessitaríem solucionar un logaritme discret:

$$\log_g(g^b) = b \;\Rightarrow\; (g^a)^b = g^{ab} = k_{AB}.$$

Es conjectura que la hipòtesi de Diffie-Hellman i la hipòtesi que el logaritme discret és intractable són equivalents.

Protocol de ElGamal

El protocol criptogràfic de ElGamal queda definit si coneixem els algorismes d'encriptació i desencriptació que es fan servir (E, D).

Sigui G un grup cíclic d'ordre n i g un generador. Sigui m el missatge que volem enviar. Per crear la clau d'aquest protocol criptogràfic es pren a aleatori, on la component pública de la clau serà $k_p = (n, g, A = g^a)$ i la component secreta $k_s = a$. Llavors, prenem un element aleatori r que farem servir per encriptar i que es mantindrà en secret. Les funcions per encriptar i desencriptar són les següents:

$$E(k_p, m) = (g^r, m \cdot A^r) =: (y_1, y_2),$$
$$D(k_s, (y_1, y_2)) = y_2 \cdot (y_1^a)^{-1}.$$

Per tant, m queda camuflat al multiplicar per A elevat a un element aleatòri, donant aleatorietat a y_2. Llavors, l'única manera de desencriptar és conèixer g^{ar} sense saber ni a ni r, només g^a i g^r. Per tant, suposant certa la hipòtesi de Diffie-Hellman el sistema és segur.

Veiem que al desencriptar el xifrat recuperem el missatge original:

$$y_2 \cdot (y_1^a)^{-1} = m \cdot A^r \cdot ((g^r)^a)^{-1} = m \cdot (g^a)^r \cdot ((g^r)^a)^{-1} = m.$$

2. Algorismes de Firma Digital

La firma digital aporta autenticitat i no-rebuig als missatges. Per aconseguir-ho es procedeix de la següent forma. Es firma el missatge amb la clau secreta de manera que només el pugui firmar una persona i es verifica la firma amb la clau pública de manera que tothom la pugui verificar.

Donat un missatge m i una parella de claus (k_p, k_s) es firma el missatge usant com a paràmetres k_s i el propi m. Posteriorment, es comprova que la firma procedeix de l'usuari propietari de la clau (k_s, k_p) usant k_p. Per dur a terme aquest procediment hi ha diferents algorismes de generació de la firma digital, amb el seu corresponent algorisme de verificació. A la parella d'algorismes se l'anomena esquema de firma digital. A continuació expliquem alguns d'aquests esquemes amb detall.

Observem primer de tot, que en la majoria de casos no es firma el missatge m, ja que aquest pot ser molt gran. Sinó que s'aplica una funció de hash al missatge i es firma $h(m) = h$ (de mida petita fixada).

Segons els paràmetres de l'algorisme de verificació podem separar els esquemes de firma digital en dos tipus: esquemes de firma digital amb apèndix i esquemes de firma digital amb recuperació de missatge.

Definició 5.2. *Esquemes de firma digital*

Esquemes de firma digital amb apèndix: necessiten el missatge com a paràmetre de l'algorisme de verificació.

Esquemes de firma digital amb recuperació de missatge: el missatge original no és necessari per verificar la firma i, encara més, el podem recuperar a partir de la firma.

Cal mencionar que per tal de recuperar el missatge amb aquest segon tipus d'esquema de firma s'ha de firmar el missatge sencer i no el hash. Per tant, aquests esquemes són molt menys pràctics.

En aquesta secció explicarem els algorismes de la firma digital relacionats amb el logaritme discret. L'únic algorisme que es coneix per falsejar la firma digital passa per trobar la clau secreta k_s, que passa per solucionar un logaritme discret. Això té cost conjecturalment exponencial per grups generals i subexponencial per grups multiplicatius d'un cos. Així doncs, la firma digital és segura.

2.1. Esquema de firma digital: Versió DSA. L'algorisme del DSA s'aplica sobre grups multiplicatius d'un cos.

L'algorisme consisteix a triar un primer q de 160-bits (per fer-ho es prenen enters aleatoris i s'aplica un test de primalitat fins a trobar un nombre primer). Es tria un altre primer $p \equiv 1 \pmod{q}$ de 512-bits. Observem que fent tries aleatòries és suficientment senzill trobar un nombre primer, ja que entre 1 i n n'hi ha aproximadament $\frac{n}{\log n}$. Llavors, el grup al que apliquem l'algorisme és $\mathbb{F}_p^*$.

Per generar la clau (k_s, k_p) primer calculem un generador g d'un subgrup de $\mathbb{F}_p^*$ de q elements. Prenem un element del grup g_0 i generem el subgrup cíclic d'ordre q que tindrà per generador $g = g_0^{(p-1)/q} \bmod p$. Per últim, prenem un enter aleatori a i calculem $y = g^a$. Llavors, $k_p = (p, q, g, y)$ i $k_s = a$.

Generació de la firma digital:

1. Es tria un enter aleatori $0 < k < q$ i es calcula $r = g^k \pmod{q}$.
2. Es computa el hash de m: $h(m) = h$.
3. Es calcula $k^{-1} \pmod{q}$.
4. Es calcula $s = k^{-1}(h + ar) \bmod q$.
5. La signatura de m és el parell (r, s).

Verificació de la firma digital:

1. Comprovar que $0 < r < q$ i que $0 < s < q$, sinó rebutjar la firma.
2. Es computa el hash de m: $h(m) = h$.
3. Es calcula $u_1 = s^{-1}h \pmod{q}$ i $u_2 = s^{-1}r \pmod{q}$.
4. Es calcula $v = g^{u_1}y^{u_2} \pmod{q}$ i s'accepta la firma si $v = r$.

Veiem que la igualtat $v = r$ es compleix pel cas de tenir una firma correcte:

(r, s) és una firma correcte $\Rightarrow\ h = ks - ar \pmod{q} \Rightarrow$ multiplicant als 2 costats per s^{-1} obtenim $k = s^{-1}h - as^{-1}r \pmod{q}\ \Rightarrow\ u_1 + au_2 = k \pmod{q}\ \Rightarrow$ $g^{u_1}y^{u_2} = g^k \pmod{q}\ \Rightarrow\ v = r$.

Observem que l'algorisme treballa amb el logaritme discret sobre un subgrup $G = \langle g \rangle \subset \mathbb{F}_p^*$. Tot i que el grup tingui ordre q la complexitat de calcular el logaritme discret depèn de p si s'aplica l'algorisme de càlcul d'índex, que és el més eficient que es coneix per aquests grups.

2.2. Esquema de firma digital: Versió ECDSA. En aquest apartat es veu la versió del DSA aplicat a corbes el·líptiques. Sigui $E(\mathbb{F}_p^*)$ una corba el·líptica.

Prenem un punt P de la corba i notem per G el subgrup generat per P d'ordre n.

Per generar la clau (k_s, k_p) procedim de la següent manera. Es prenen com a component pública $k_p = (p, n, P, Q)$, on $Q = aP$ amb a aleatori. Com a component privada es pren $k_s = a$.

Generació de la firma digital:

1. Es pren un enter $0 < k < n$ i es calcula $kP = (x_1, x_2)$.
2. Convertim x_1 a enter i computem $r = x_1 \pmod n$.
3. Computem el hash del missatge $h(m) = h$.
4. Es computa $s = k^{-1}(h + ar) \pmod n$.
5. La signatura de m és el parell (r, s).

Verificació de la firma digital:

1. Es comprova que $0 < r < n$ i que $0 < s < n$, sinó rebutjar la firma.
2. Es calcula $u_1 = s^{-1}h \pmod n$ i $u_2 = s^{-1}r$.
3. Es computa $X = u_1P + u_2Q = (x_2, y_2)$, si $X = P_\infty$ rebutjar la firma.
4. Es converteix x_2 a enter i computem $v = x_2 \pmod n$.
5. S'accepta la firma si $v = r$

Veiem que si tenim una firma correcta l'algorisme de verificació l'accepta: $(x_2, y_2) = X = u_1P + u_2Q = (u_1 + u_2a)P = (s^{-1}h + s^{-1}ar)P = (s^{-1}\underbrace{(h + ar)}_{ks})P = kP = (x_1, y_1)$. Per tant, tenim que $v = x_2 \pmod n = x_1 \pmod n = r$.

2.3. Esquema de firma digital de Nyberg-Rueppel. Aquest esquema és un esquema amb recuperació de missatge, per tant, en l'algorisme de verificació no prenem el missatge com a paràmetre. S'agafa un primer p i llavors $\mathbb{F}_p^*$ és l'espai que es fa servir per generar l'algorisme.

Per trobar la clau (k_s, k_p) es procedeix de la mateixa forma que en l'algorisme DSA. Es pren un primer q que divideixi $(p - 1)$. S'agafa un element del grup g_0 i es crea el subgrup cíclic d'ordre q amb generador $g = g_0^{(p-1)/q} \bmod p$. Per últim, es pren un enter aleatori a i es calcula $y = g^a$. Les components de la clau

són $k_p = (p, q, g, y)$ i $k_s = a$.

En el cas dels esquemes amb message recovery necessitem una funció de redundància R.

DEFINICIÓ 5.3. *Funció de redundància (Redundancy function)*
Una funció de redundància és una funció de la següent forma:

$$\begin{array}{rccc} R: & \{0,1\}^\ell & \longrightarrow & \mathcal{M}_R \\ & m & \longmapsto & R(m) = \widetilde{m} \end{array}$$

On el conjunt $\mathcal{M}_R$ compleix que trobar una firma digital s tal que l'algorisme de verificació obtingui un element de $\mathcal{M}_R$ és intractable.

Veiem un exemple de com construir $\mathcal{M}_R$ i R on és improbable que una cadena binaria aleatòria B compleixi $B \in \mathcal{M}_R$:

$$\mathcal{M}_R = \{\widetilde{m} \in \{0,1\}^{2\ell} \mid \widetilde{m} = m||m,\ m \in \{0,1\}^\ell\}.$$

On $||$ simbolitza concatenar. Per tant, la probabilitat de que un missatge aleatòri estigui en $\mathcal{M}_R$ és $\left(\frac{1}{2}\right)^\ell$, ja que l'element el bit i-èssim ha de coincidir amb el bit $2i$-èssim per $i \in \{1, ..., \ell\}$.

Generació de la firma digital:

1. Es computa $\widetilde{m} = R(m)$.
2. Es selecciona un enter aleatòri $0 < k < q$ i es calcula $r = g^{-k} \pmod p$.
3. Es computa $e = \widetilde{m}r \pmod p$ i $s = ae + k \pmod q$.
4. La signatura de m és el parell (e, s).

Verificació de la firma digital:

1. Comprovar que $0 < e < p$ i que $0 < s < q$, sinó rebutjar la firma.
2. Es computa $v = g^s y^{-e} \pmod p$ i $\widetilde{m} = ve \pmod p$.
3. Es comprova que $\widetilde{m} \in \mathcal{M}_R$, sinó es rebutja la firma.
4. Es recupera el missatge $m = R^{-1}(\widetilde{m})$.

Veiem que l'algorisme de verificació accepta la firma generada per l'esquema:

$$v = g^s y^{-e} = g^{s-ea} = g^{ae+k-ea} = g^k = r^{-1}.$$

Llavors, $v \cdot e = r^{-1}\widetilde{m}r = \widetilde{m} \in \mathcal{M}_R$. Per recuperar el missatge només cal invertir la funció R obtenint $m = R^{-1}(\widetilde{m})$.

Bibliografia

[1] Neal Koblitz, A Course in Number Theory and Cryptography, Springer-Verlag 2nd edition, (1994). *Secció 4.3 index calculus i Capítol 6 elliptic curves.*

[2] Neal Koblitz, Algebraic Aspects of Criptography, Springer, (1998). *Capítols 5 i 6 elliptic curves.*

[3] Song Y. Yan, Number theory for Computing, Springer 2nd edition, (2002). *Seccions 1.7 arithmetic of elliptic curves, 2.4 Shank's algorithm, Silver-Pohlig-Hellman.*

[4] Alfred J. Menezes, Paul C. van Oorschot i Scott A. Vanstone, Handbook of Applied Cryptography, CRC Press 5th edition, (2001). *Capítol 9.*

[5] Chris Studholme, The Discrete Log Problem, (June 21, 2002). [http://www.cs.utoronto.ca/∼cvs/dlog/].

[6] Alfred J.Menezes, Elliptic Curve public key cryptosystems, Kluwer Academic Publishers, (2001).

[7] Victor Shoup, Lower bounds for discrete logarithms and related problems, (1997). [www.shoup.net/papers/dlbounds1.pdf].

[8] Stephen D. Miller and Ramarathnam Venkatesan, Non-degeneracy of Pollard Rho Collisions, (August 4, 2008). [http://arxiv.org/abs/0808.0469].

[9] Adolf Hildebrand and Gerald Tenenbaum, On integers free of large prime factors, Transactions of the American Mathematics Society 296, *pàg. 265-290* (1986).

[10] Oliver Schirokauer, Discrete logarithms and local units, Philosophical Transactions: Physical Sciences and Engineering 345, *pàg. 409-423* (1993).

[11] Paul E. Black, Dictionary of Algorithms and Data Structures, U.S. National Institute of Standards and Technology (July 17, 2006).

Apèndix A
Funció de Hash (SHA-1)

A continuació descriurem l'algorisme SHA-1 (Secure Hash Algorithm 1) que va proposar l'U.S. National Institute of Standards and Technology (NIST). Va ser el primer algorisme per crear funcions de hash criptogràfiques segures. Actualment, està desfasat ja que s'han trobat maneres d'obtenir col·lisions i comprometre'n la seguretat. Les versions posteriors de l'algorisme són SHA-2 i SHA-3.

Per l'algorisme de SHA-1 necessitem introduir una serie de notacions sobre les operacions que farem servir. Notarem les operacions bit a bit XOR com $\oplus$, AND com $\wedge$, OR com $\vee$ i NOT com $\neg$.

$\oplus$	0	1
0	0	1
1	1	0

$\wedge$	0	1
0	0	0
1	0	1

$\vee$	0	1
0	0	1
1	1	1

$\neg$	
0	1
1	0

Algorisme A.1. SHA-1

Data: Missatge m de longitud ℓ arbitràriament gran.

Result: El hash del missatge h de 160-bits.

Definim les variables de 32-bits següents:

$h_0 = 67452301,\ h_1 = EFCDAB89,\ h_2 = 98BADCFE,\ h_3 = 10325476,\ h_4 = C3D2E1F0$;

Precomputació:

Afegir un 1 al final del missatge.

Afegir zeros fins longitud $\equiv 448 \pmod{512}$.

Afegir 64-bits corresponents a $\ell \pmod{2^{64}}$.

Obtenim $\overline{m} = (m, 1, 0, ..., 0, \ell)$.

Computació:

Tallar $\overline{m}$ en cadenes de 512-bits.

for *cada cadena* **do**

Trenquem els 512-bits en 16 cadenes de 32-bits: $w_1, ..., w_{16}$.

Creem noves cadenes de 32-bits (fins a 80):

$w_i = w_{i-3} \oplus w_{i-8} \oplus w_{i-14} \oplus w_{i-16}$ i després moure cada bit una posició a l'esquerre en cicle. Obtenint: $w_{17}, ..., w_{80}$.

$a = h_0,\ b = h_1,\ c = h_2,\ d = h_3,\ e = h_4$;

for $i = 0$ *to* 80 **do**

From 1 to 20: $f = (b \wedge c) \vee ((\neg b) \wedge d),\ k = 5A827999$;

From 21 to 40: $f = b \oplus c \oplus d,\ k = 6ED9EBA1$;

From 41 to 60: $f = (b \wedge c) \vee (b \wedge d) \vee (c \wedge d),\ k = 8F1BBCDC$;

From 61 to 80: $f = b \oplus c \oplus d,\ k = CA62C1D6$;

$\widetilde{a} = a$ movent cada bit 5 llocs cap a l'esquerre en cicle;

$\text{temp} = \widetilde{a} + f + e + k + w_i \pmod{2^{32}}$;

$e = d,\ d = c,\ c = b$ (mogut 30 a l'esquerre), $b = a,\ a = \text{temp}$;

end

$h_0 = h_0 + a \pmod{2^{32}};\ h_1 = h_1 + b \pmod{2^{32}};\ h_2 = h_2 + c \pmod{2^{32}};$

$h_3 = h_3 + d \pmod{2^{32}};\ h_4 = h_4 + e \pmod{2^{32}};$

end

return $h = (h_0, h_1, h_2, h_3, h_4)$;

Observem que l'algorisme passa d'un missatge de longitud arbitràriament llarga a un hash de 160-bits. Això ens permet tractar la integritat d'un missatge de manera molt menys costosa, ja que només hem d'assegurar integritat pels 160 bits del hash.

Apèndix B
Implementació dels algorismes

Algorisme per al càlcul de l'exponencial discreta

Algorisme B.1. $\exp(g, k)$
Data: g `un element del grup,` k `un enter.`
Result: $\exp(g, k) = g^k$.
if $k = 1$ **then**
  return $\exp(g, k) = g$;
else if k *parell* **then**
  return $\exp(g, k) = \exp(g, k/2) \cdot \exp(g, k/2)$;
else if k *senar* **then**
  return $\exp(g, k) = \exp(g, (k-1)/2) \cdot \exp(g, (k-1)/2) \cdot g$;
end

Algorisme de força bruta

Algorisme B.2. $\log(g, n, h)$

Data: g `generador del grup,` $n = |G|$`,` h `un element de` G`.`

Result: $\log(g, n, h) = \log_g(h)$.

$A = 1$;

for $k = 0$ *to* $n - 1$ **do**

  if $A = h$ **then**

    `break;`

  else

    $A = g \cdot A$;

  end

end

return k;

Algorisme per calcular múltiples logaritmes amb força bruta

Algorisme B.3. `Precomputació`

Data: g `generador del grup,` $n = |G|$.

Result: `Taula ordenada de totes les exponencials de` G.

$A[0] \;=\; (1, 0)$;

for $k = 1$ *to* $n - 1$ **do**

$\quad A[k] \;=\; (g \cdot A[k-1][1],\;\; k)$;

end

`Sort(A) - Ordenem A per la primera component;`

Algorisme B.4. $\log(g, n, h)$

Data: g `un generador del grup,` $n = |G|$, h `un element del grup` G.

Result: $\log(g, n, h) \;=\; \log_g(h)$.

$d = n$ `Posició de la dreta del vector.`

$e = 1$ `Posició de la esquerra del vector.`

`Cerca_Dicotòmica(A,e,d,h):`

$m \;=\; \text{round}\left(\dfrac{e+d}{2}\right)$;

if $A[m] < h$ **then**

$\quad \log(g, n, h) \;=$ `Cerca_Dicotòmica(A,m,d,h);`

else if $A[m] > h$ **then**

$\quad \log(g, n, h) \;=$ `Cerca_Dicotòmica(A,e,m,h);`

else if $A[m] = h$ **then**

$\quad \log(g, n, h) \;=\; m$;

end

Algorisme de Shanks

Algorisme B.5 (Shanks)**.**

Data: g un generador del grup, $n = |G|$, h un element de G.
Result: $\log_g(h) = k$.
Computem $s = \lfloor \sqrt{n} \rfloor$;
Baby-step:
Computem i guardem $S = \{(g^r, r) \mid r \in \{0, 1, 2, ..., s-1\}\}$;
Sort(S) - respecte a la primera component;
Giant-step:
for $q = 0$ *to* $s + 1$ *and not match* **do**
    Computem $(h \cdot g^{-sq}, q)$;
    Busquem un match en S. (usar Cerca_Dicotòmica)
end
return $k = r + sq$;

Algorisme ρ de Pollard

Algorisme B.6 (Pollard)**.**

Data: g `un generador del grup,` $n = |G|$, h `un element de` G.
Result: $\log_g(h) = k$.
$x_1 = g,\ \ a_1 = 1,\ \ b_1 = 0$;
$(x_1, a_1, b_1) =$ `walk`(x_1, a_1, b_1);
$(x_2, a_2, b_2) =$ `walk`(x_1, a_1, b_1);
while $x_1 \neq x_2$ **do**
    $(x_1, a_1, b_1) =$ `walk`(x_1, a_1, b_1);
    $(x_2, a_2, b_2) =$ `walk`$($`walk`$(x_2, a_2, b_2))$;
end
if $b_1 \equiv b_2\ (mod\ n)$ **then**
    return `Solució no trobada;`
else
    return $(a_1 - a_2)(b_2 - b_1)^{-1}\ \ mod\ \ n$;
end

Algorisme del Cangur

Algorisme B.7 (Kangaroo)**.**

Data: w la mida de l'interval, g un generador del grup, $n = |G|$, $h \in G$, D el conjunt de punts distingits.

Result: $\log_g(h) = k$.

$x = g^{\lfloor w/2 \rfloor}$, $a = \lfloor w/2 \rfloor$;

$y = h$, $b = 0$;

$S_1 = S_2 = \emptyset$;

while *no hem trobat el match i no estem en un cicle* **do**

  $(x, a) = \texttt{walk}(x, a)$;

  $(y, b) = \texttt{walk}(y, b)$;

  if $x \in D$ **then**

    $S_1 = S_1 \cup \{(x, a)\}$; - Col·locar de forma ordenada

    $x \in S_2$; - Hem trobat el match

    $x \in S_1$; - Hem trobat un cicle

  end

  if $y \in D$ **then**

    $S_2 = S_2 \cup \{(y, b)\}$; - Col·locar de forma ordenada

    $y \in S_1$; - Hem trobat el match

    $y \in S_2$; - Hem trobat un cicle

  end

end

return $k = a - b$;

Trobarem el match quan el cangur que està per darrere arribi a un punt distingit, passada la primera col·lisió.

Per evitar que l'algorisme entri en un bucle infinit imposem la condició que les seqüències no entrin en un cicle abans del match. En aquest cas, l'algorisme pot fallar pel fet que un dels cangurs vagi repetint salts pels que l'altre cangur no passa mai.

Algorisme de Trial Division

Algorisme B.8. `Trial Division`

Data: `Un element del grup` $h \in G$, `una fita` B `i la base de factors` $\mathcal{B} = \{p_1, ..., p_r\}$.

Result: `Factorització de` $h = p_1^{k_1} \cdot ... \cdot p_r^{k_r}$.

`Definim:` $k_1 = k_2 = ... = k_r = 0$;

`Definim:` $a = h,\ b = p_1$;

`Definim:` $i = 1,\ p_{r+1} = B + 1$;

while $a \neq 1$ *or* $b \leqslant B$ **do**

  $(q, r) =$ `Divide`(a, b); `- Dividim a entre b,` $q =$ quocient, $r =$ residu.

  if $r = 0$ **then**

    `//Divisió exacta - Continuem amb el mateix factor`

    $a = q$; `//Seguim dividint el quocient`

    $k_i = k_i + 1$; `//Afegim 1 a l'exponent de` p_i

  else if $r \neq 0$ **then**

    `//Divisió no exacta - Passem al següent factor`

    $i = i + 1$;

    $b = p_i$;

  end

end

return $(k_1, ..., k_r)$;

Algorisme Polynomial Sieve

Algorisme B.9. `Polynomial Sieve`

Data: `Un polinomi` $f(x) = a_n x^n + ... + a_1 x + a_0$, `un interval` $[c, d)$ `amb` $c \leqslant x < d$ `i la base de factors` $\mathcal{B} = \{p_1, ..., p_r\}$.
Result: `Ens diu quins elements són llisos i quins no ho són.`
for $c \leqslant x < d$ **do**
  $l(x) = 0$; `//logaritme dels primers que divideixen` $f(x)$
end
for *all* $p \in \mathcal{B}$ **do**
  `Definim:` $f_p(x) = f(x) \pmod p$; `//Coeficients de` $f(x)$ `mòdul` p
  `Definim:` $z_p = \{z \mid f_p(z) \equiv 0 \pmod p$ `amb` $0 \leqslant z < p\}$
  if $f_p \equiv 0 \pmod p$ **then**
    for $c \leqslant x < d$ **do**
      $l(x) = l(x) + \log(p)$;
    end
  else
    for *all* $z \in z_p$ **do**
      `Definim:` $x_p = -c + z \pmod p$;
      `Definim:` $x = c + x_p$;
      while $x < d$ **do**
        $l(x) = l(x) + \log(p)$;
        $x = x + p$; `//`$f(x+p)$ `també divisible per` p `si` $f(x)$ `ho és`
      end
    end
  end
  `//Comprovem divisibilitat per potències de` p
  for $z_0 \in z_p$ **do**
    `//Suposem` z_0 `divisible per` p^{e-1} `i mirem si divisible per` p^e
    `Definim:` $f_{p^e}(W) = f(z_0 + Wp^{e-1})/p^{e-1} \pmod p$;
    `Definim:` $w_{p^e} = \{w \mid 0 \leqslant w < p$ `tal que` $f_p(w) \equiv 0 \mod p\}$;
    $z_{p^e} = \{z \mid 0 \leqslant z < p$ `tq` $z \equiv z_0 + wp^{e-1} \pmod p$ `per` $w \in w_{p^e}\}$;
    for *all* $z \in z_{p^e}$ **do**
      `Definim:` $x_p = -c + z \pmod p$;
      `Definim:` $x = c + x_p$;
      while $x < d$ **do**
        $l(x) = l(x) + \log(p)$;
        $x = x + p$; `//`$f(x+p)$ `també divisible per` p `si` $f(x)$ `ho és`
      end
      `Comprovar divisibilitat per` p^{e+1} `de la mateixa forma;`
    end
  end
  for $c \leqslant x < d$ **do**
    if $l(x) \approx \log(x)$ **then**
      `Llavors:` $f(x)$ `és un element llis;`
    end
  end
end

Per a aquest algorisme cal fer notar que si $f(b)$ és divisible per un primer p, llavors $f(b+p)$ també és divisible per aquest primer. A més, si deinim $f_p(x) = f(x) \pmod{p}$, on prenem cada coeficient del polinomi mòdul p. Llavors, $f(b)$ és divisible per $p \Leftrightarrow f_p(b)$ és divisible per p.

L'algorisme fa servir logaritmes a l'hora de fer la comparació per decidir si un element és llis, ja que el cost de fer multiplicacions en un ordinador és superior que el de sumar. En aquest treball no considerem la diferència de cost de les operacions bàsiques, suposem que tenen cost constant. Tot i així, es pot veure que aquest algorisme serà més eficient si ho fem d'aquesta forma.

Un segon fet important a tenir en compte és que l'algorisme B.9 pot fallar amb baixa probabilitat (per errors d'aproximació en la funció $l(x)$). Tant pot dir que un element que no és llis sí que ho és com al revés. Aquest fet no és massa transcendent en el cas que volem tractar, ja que després buscarem els primers en que factoritza cada element obtingut pel mètode (usant Trial Division) amb el que veuríem realment si l'element en qüestió era llis.